PRACTICAL PROGRAMMING IN BASIC
Study Unit 7
24707-2
Ed 1

COMPUTER LOGIC - THE LIMITS OF COMPUTER INTELLIGENCE

24707-2/8507 Printed in South Africa

SCHOOL OF COMPUTER TRAINING

PROGRAMMING IN BASIC
STUDY UNIT 7

COMPUTER LOGIC— THE LIMITS OF COMPUTER INTELLIGENCE

(A4707) © Copyright 1983/Intext, Inc., Scranton, Pennsylvania 18515
Printed in the United Sates of America

Edition 2

STUDY UNIT 7

YOUR LEARNING OBJECTIVES

WHEN YOU COMPLETE THIS UNIT, YOU WILL BE ABLE TO:

□ Use your computer to compare two values. This is the basis of "artificial intelligence" **Pages 1–2**

□ Determine the relationship between two values by using the keyword IF .... **Pages 3–4**

□ Understand how, in a numeric comparison, values are compared according to common sense .... **Pages 4–5**

□ Compare alphanumeric or string data using the rules of the computer's collating sequence .... **Pages 5–8**

□ Use a business payroll program to demonstrate how the IF statement can be used to vary the way records are processed .... **Pages 10–13**

□ Follow the alternate paths taken when branching occurs via an IF statement by walking through the sample program .... **Pages 14–19**

□ Test several conditions in one IF statement through the use of the logical operators: AND and OR .... **Page 26**

□ Make the payroll program more realistic by adding such logic as bonus pay when a few modifications are made .... **Appendix**

LEARNING AIDS
Programmer's Check #1 ..... **8–9**
Programmer's Check #2 .. **20–21**
Programmer's Check #3 .. **28–29**

EXAM 7 (Examination for Study Unit 7)........... **39–42**

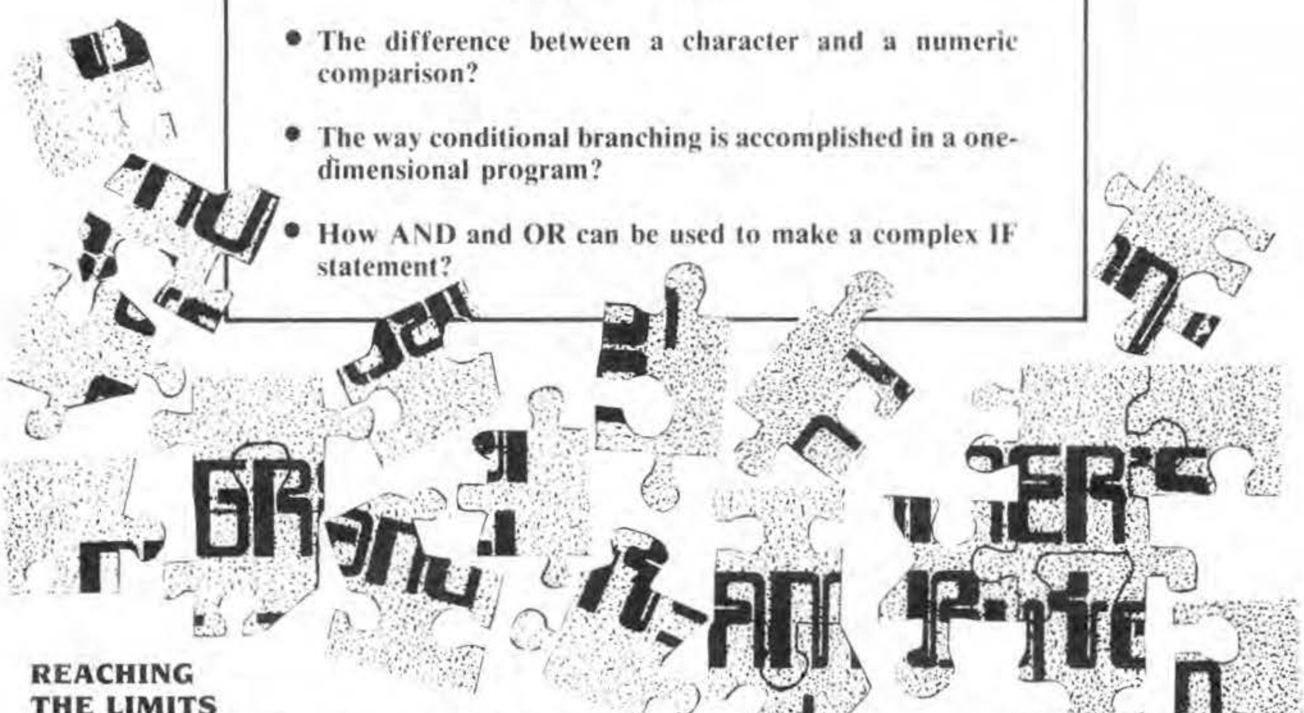
ANSWER SHEET **43**

STUDY UNIT 7

COMPUTER LOGIC — THE LIMITS OF COMPUTER "INTELLIGENCE"

DO YOU KNOW?

- **The difference between a character and a numeric comparison?**
- **The way conditional branching is accomplished in a one-dimensional program?**
- **How AND and OR can be used to make a complex IF statement?**



REACHING THE LIMITS

There is a great deal of satisfaction and security in knowing that things have limits and boundaries. Imagine the frustration, for example, if you were attempting to put together a jigsaw puzzle which had an unlimited number of pieces. How long would you grapple with the pieces before giving up?

On the other hand, when you open a box containing a 5,000 piece puzzle, you have satisfaction in knowing it is possible to complete the puzzle. The number of pieces is finite. All you need to do is be patient, concentrate on identifying the colors, shapes and sizes and, eventually, you can put the whole picture together.

When approaching a puzzle, certain routines and conventions are followed. You don't randomly pick up a piece and attempt to match it with all the other pieces. Usually, you spread all the pieces out on a table with the picture-side up. Then, you quickly identify all the border pieces and begin assembling them.

You know the border pieces are easily identified. Most puzzle fans take great satisfaction in getting the border together fast! And once the border is completed, there is more satisfaction in knowing that the remaining pieces will fit inside the border.

What steps are next? Well, this is where puzzle solution styles diverge. Some experts attack the remaining pieces by color and picture features, gathering pieces into groups before attempting to put the elements together. Others attack the puzzle pieces by concentrating on one feature of the picture such as a mountain or boat or wall.

Another school of puzzlers immediately attempt to fit pieces into the inside of the border without additional fanfare or other preparation. This process eventually reaches the same objective as all other approaches: the puzzle is completed.

Learning how to design computer programs is very much like finding the best ways to complete what at first may appear to be a puzzle. You have no doubt realized by this time that one person's style of solving a puzzle or creating a program can be different from another person's, provided that the result is successful.

You should be enjoying a certain amount of satisfaction by now, knowing that the "intelligence" of a computer is finite. Like any good puzzle, there are boundaries and limits to the numbers and types of pieces which can fit properly. And, you are in the process of learning how best to approach program puzzles so you can complete designs on paper that will, when tested, work in the computer. That, finally, is the ultimate satisfaction for the programmer: seeing the completed picture come together instantly on the CRT once the last piece of the puzzle is entered.

This Study Unit will explore some of the limitations of the computer and how it functions.You will learn how the computer helps you to sort out numeric and character data. You will discover the logical processes used by the computer in problem solving. Finally, you will see how the computer, itself, works out certain parts of the program before moving on toward putting the picture together. How does the computer logically compare one piece of the puzzle with another? Quite simply, the size of each puzzle piece is one of the computer's means, as you will see in the discussion which follows.

COMPARING

The capability of comparing is what we usually mean by computer intelligence. The Central Processing Unit (CPU) allows us to compare two values and to have one of these possible results given:

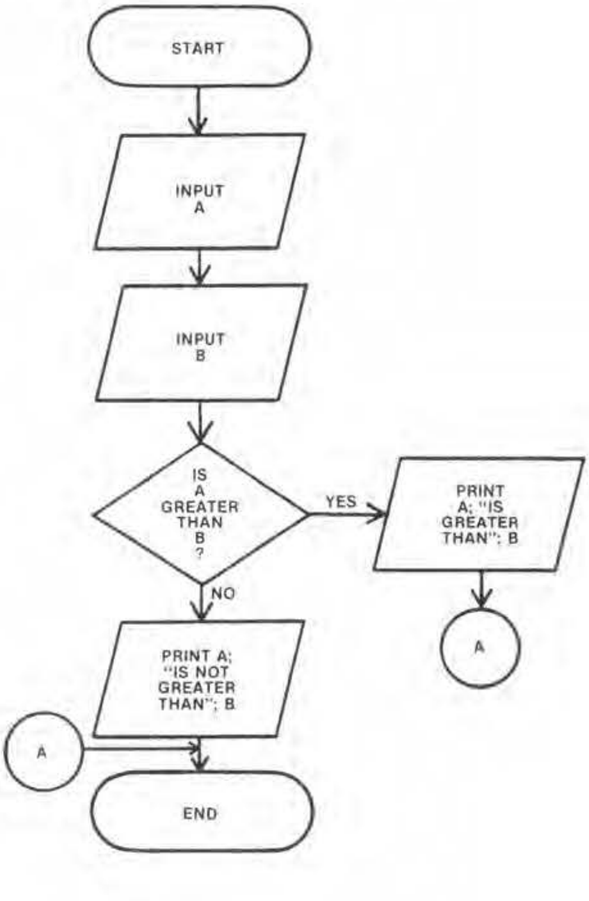
(A) The two values are equal.

(B) The first value is less than the second value.

(C) The first value is greater than the second value.

The result of the comparison may then be used to control the direction that our program will take. We are able to code different logic into a program so that it can handle data differently from another program.

For example, a program can be written to compare two given numbers. The display will show different lines of output, depending on the results.


START
INPUT A
INPUT B
IS A GREATER THAN B ?
YES
PRINT A; "IS GREATER THAN"; B
A
NO
PRINT A; "IS NOT GREATER THAN"; B
A
END

THE "IF" STATEMENT

In BASIC, the keyword IF is used to make a comparison. In an IF statement, a comparison between two variables or a variable and a constant is made. If the statement turns out to be "false," the next instruction in the program is performed. The second part of the IF statement is the THEN clause. The THEN clause indicates what to do if the comparison turns out to be "true."

The format of the IF statement is:

$$\textbf{nn IF}\begin{Bmatrix}\textbf{literal or}\\ \textbf{variable}\end{Bmatrix} \quad \begin{Bmatrix} > & >= \\ < & <= \\ = & >< \end{Bmatrix}$$

$$\begin{Bmatrix}\textbf{literal or}\\ \textbf{variable}\end{Bmatrix}\ \textbf{THEN}\ \begin{Bmatrix}\textbf{imperative}\\ \textbf{statement}\end{Bmatrix}$$

where: nn is the line number.

IF is the keyword.

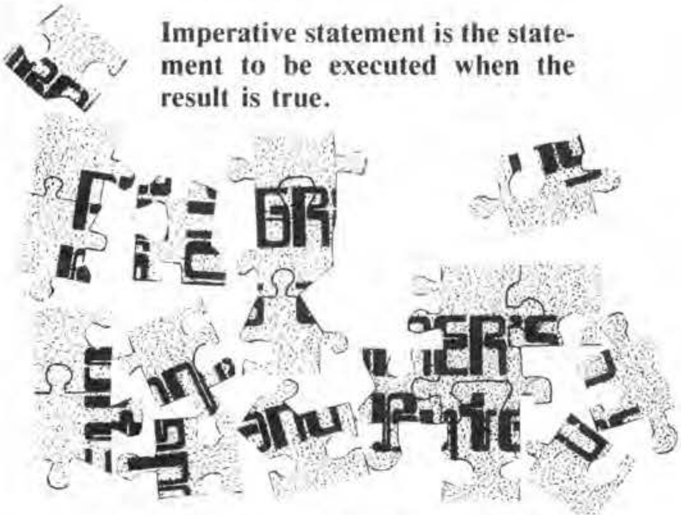
***Literal or variable* contain the first and second values to be used in the comparison.**

>, <, =, >=, <=, ><

are the comparing operations.

THEN indicates what to do if the result is true.

Imperative statement is the statement to be executed when the result is true.

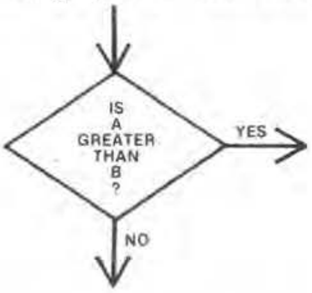


The following table gives the English translations of the comparing symbols:

GREATER THAN, EQUAL TO...
OR LESS THAN...

SYMBOL	MEANING
>	.... greater than
<	.... less than
=	.... equal to
>=	.... greater than or equal to
<=	.... less than or equal to
><	.... less than or greater than (not equal to)

In the preceding flowchart, a decision box was used to show that a logical branch should occur depending upon the values of A and B:



We could code this as:

```
3Ø IF A> B THEN...
```

However, the computer cannot accept this instruction until we complete the THEN clause. That is, we must know what we want to be done, only when the result is true (that A's value is greater than B's).

When the result is false (that B's value is greater than A's), an entirely different statement is to be done.

NOTE: The way this statement will be fully coded depends, to some extent, on the version of BASIC you are working with. Check your manual to determine if you have use of the ELSE clause. If you do, you might utilize this feature by coding statement 3Ø as follows:

```
3Ø IF A>B THEN PRINT A;
"IS GREATER THAN"; B ELSE PRINT A;
"IS NOT GREATER THAN"; B
```

Every version of BASIC, however, will work with the coding that follows.

The next statement in sequence will be: 4Ø PRINT A; "IS NOT GREATER THAN"; B. It will be executed only when the IF statement is *false*. Therefore, the steps to be performed when the result is true, must be located elsewhere in the program. Note how this is accomplished in a program solution to our previously diagrammed flowchart:

```
1Ø INPUT A

2Ø INPUT B

3Ø IF A>B THEN GOTO 6Ø

4Ø PRINT A; "IS NOT GREATER THAN";
   B

5Ø GOTO 7Ø

6Ø PRINT A; "IS GREATER THAN"; B

7Ø STOP
```

See how the IF statement on line 3Ø will cause a branch (via a GOTO statement) to line 6Ø where a line will be printed before the STOP on line 7Ø ends the run.

In this way, every time two values are entered for A and B, only one line of output will be printed (on lines 4Ø or 6Ø).

Enter this program and RUN it. When prompted, enter values for A and B. The CRT will display which of the two it found to be greater by branching to the appropriate statement. RUN it several more times. See how equal values will not cause a branch. Remember, we could not forget to consider the equal condition. Now try these values for A and B:

A = 1.5Ø
B = Ø1.5

Note that the computer will consider these two values to be equal and thus A is not greater than B.

NUMERIC COMPARISONS

When comparing numeric values, the CPU cannot make the comparison until it has first done two preliminary steps:

1. The decimal points of the two values are aligned and,

2. the fields are made equal in length by padding one or the other field (or both) with zeros.

In the previous example, the following would take place:

1. The decimal points would be aligned as:

   A = 1.5Ø

   B = Ø1.5

2. Both A and B would be padded with zeros:

   A = Ø1.5Ø

   B = Ø1.5Ø

3. The bytes would be checked from right to left, for the entire lengths of the fields.

4. A condition code is set in ROM to show the results of the comparison (equal, less than or greater than).

5. The program will then branch (or not) as directed.

To state it in plain terms, a numeric comparison follows common sense!

NOTE: The program we have just coded could have been made a little bit awkward in its structure, and, on some computers, not allowable! That is, if we had reversed lines 5Ø and 7Ø so that the STOP statement is not at the end of our program, as in the following:

```
1Ø INPUT A

2Ø INPUT B

3Ø IF A>B THEN GOTO 6Ø

4Ø PRINT A; "IS NOT GREATER THAN";
   B

5Ø STOP

6Ø PRINT A; "IS GREATER THAN"; B

7Ø GOTO 5Ø
```

Whenever possible, the STOP statement should be at the bottom of your program.

Now, let's see what would have happened if we had omitted the GOTO statement on line 5Ø. DELETE line 5Ø. RUN the program. If you now enter values for A and B, some results may be surprising. As long as A *is* greater than B, the program works fine. But once A's value is *not* greater than B's (or equal), two lines of output will be displayed.

```
1Ø INPUT A

2Ø INPUT B

3Ø IF A>B THEN GOTO 6Ø

4Ø PRINT A; "IS NOT GREATER THAN";
   B

6Ø PRINT A; "IS GREATER THAN"; B

7Ø STOP
```

So much for the computer's intelligence—without the GOTO statement, it had no idea that we wanted only one of the two lines to be printed. Be cautious whenever you use IF statements. Translating a two-dimensional flowchart into a linear program requires careful coding. Make sure that you walk through your coding designs *before* you RUN them. Remember, the computer will only do what you tell it do; it will never "figure out" what you meant to do!

FIGURE 1—Smart contractors are seeking out programmers to create reliable programs for estimating time and materials. Knowing precisely the amount of insulation required for a two-story, six-room home will save money and reduce waste. Such questions as "GREATER THAN," "LESS THAN," and "EQUAL TO" apply when estimating jobs.

Now, let's see how the computer can compare alphanumeric, or string, variables.

ALPHANUMERIC COMPARISONS

When the computer is asked to compare alphanumeric values (constants or variables which may contain non-numeric data, such as letters or special characters like commas, periods, dollar signs, etc.), it uses no common sense, at all! Ask the proverbial man-on-thestreet which is greater, the letter Z or the number 5, and you are likely to get a very puzzled expression. Then, ask someone whether a comma is less than a period and they are likely to question your sanity!

Actually, the computer is asked to answer questions like this quite often, and, its response is unambiguous. That is because every computer has a "collating sequence" wherein every possible character is assigned a value. The relative positions of the characters within the collating sequence will be found in your technical manual. Some systems use the EBCDIC system (Extended Binary Coded Decimal Interchange Code), while others use ASCII (American Standard Coding Information Interchange). We will describe the EBCDIC system here.

The lowest "printable" character in the collating sequence is a space (or a blank). That is to say, *any* character is greater than a space. The special characters are the next highest values that can be contained in a byte of main storage. Then come the letters of the alphabet—from the lowest, A, through the highest, Z.

Finally, we reach the numbers zero through nine. This collating sequence is illustrated below (see box).

To restate, any character is greater than a blank and any character is less than the number 9; numbers are greater than letters.

There is a logic to this sequence, as you can see if you examine the alphabetical listings in the phone book. The "least most" names (those starting with "A") are at the front of the book; the greatest names (those starting with Z) are at the back.

Alphanumeric comparisons are made between string variables, in BASIC, in the same way that numerics are compared. We can try one in the following example:

```
1Ø INPUT A$

2Ø INPUT B$

3Ø IF A$>B$ THEN GOTO 6Ø

4Ø PRINT A$; "IS NOT GREATER
   THAN"; B$

5Ø GOTO 7Ø

6Ø PRINT A$; "IS GREATER THAN";
   B$

7Ø STOP
```

Each time that you RUN this program, you will find proof of the collating system your computer uses. Try running the program with the following values for A$ and B$. Then note the results in the space provided:

	A$	**B$**	**RESULTS (A$> B$ or A$≤B$)**
(A)	**A**	**B**	
(B)	**Z**	**A**	
(C)	**C**	**(space)**	
(D)	**7**	**8**	
(E)	**Q**	**Q**	

LOWEST PRIORITY.....TO——————————→ HIGHEST PRIORITY

ƀ	; , . ?	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Ø123456789
(blank space)	special characters	letters	numbers

You should find that the further the letter is in the alphabet, the greater its value and that, when single-digit numbers are entered, we find the same results as we would expect from a numeric comparison.

More revealing results, however, can be found by entering a letter and a number. Try these:

	A$	B$	RESULT
(F)	1	A	
(G)	P	6	

If your system uses EBCDIC, you should find that in (F) A$ is greater than B$ and in (G) that A$ is not greater than B$. If the ASCII system is used, the results will be exactly the *opposite*.

Running the program with these values will provide some surprising conclusions:

	A$	B$	RESULT
(H)	1.1Ø	1.1	
(I)	.1	Ø.1	

Our own common sense tells us that, in both of the above cases, the two values should be equal. Indeed, if this was a numeric comparison, the computer would find them to be equivalent. But that is because, in a numeric comparison, decimal alignment and padding with zeros occurs before the fields are compared. In an alphanumeric comparison, the rules are quite different.

When string values are compared, the following steps take place:

1. If the fields are not equal in *length*, the shorter field is padded with blanks (spaces) on the *right* side.

2. The fields are compared, byte by byte from left to right.

3. The comparison ends when one of the following occurs:

   (A) The comparison continues to the rightmost byte and all of the values are equal. Then, the two sets of characters are said to be *equal*.

   (B) An unequal pair of bytes are encountered. The comparison immediately ends and the field with the greater values in that byte is said to be greater.

Let's see how these rules can be applied to make some sense out of our previous two examples.

In the first case, A$ (1.1Ø) is four bytes long; B$ (1.1) is three. In a numeric comparison, the decimal points would be aligned and B$ would be filled with a Ø. The two values would be considered equal.

A$ = 1.1Ø

$$B\$ = 1.\underbrace{1\emptyset}_{\text{Padded}}$$

But, in the alphanumeric comparison, B$ is padded with a blank:

A$	=	1	.	1	Ø
B$	=	1	.	1	ƀ
					↑ (space)

As the comparison proceeds from left to right, the first three pairs of bytes are found to be equal (1.1). The fourth byte, however, is not equal. A "Ø" is greater than a blank; therefore, A$ is greater than B$.

In the next example, A$ (.1) is not greater than B$ (Ø.1) because Ø is greater than a period:

			padded	
A$	=	.	1	ƀ
B$	=	Ø	.	1

An important lesson can be learned from this. When comparing numeric values, make sure that the fields are defined as numeric; otherwise, the output may be undependable.

Now, let's compare these names:

	A$	B$	RESULT
(J)	Sam	Tom	________________
(K)	Mark	Marc	________________
(L)	Carl	Carla	________________

In (J), A$ is not greater than B$ because the first byte of A$ (S) is not greater than the first byte of B$ (T). In (K), A$ is greater than B$, because, in the fourth byte, K is greater than C.

In the third example, (L), A$ is not greater than B$ because, in the fifth byte, A is greater than the blank the first variable is padded with.

String variables may also be compared to constants as long as the constant is enclosed within quotation marks. The following BASIC statements are often used to provide an escape from a loop:

```
4Ø INPUT A$
5Ø IF A$ = "N" THEN GOTO 7Ø
6Ø GOTO 4Ø
7Ø STOP
```

Remember, when comparing to a literal, the use of quotation marks entirely depends on the definition of the variable (*not* the literal itself). Therefore, both of these BASIC statements are correctly coded:

```
1Ø IF A>5 THEN GOTO 5Ø
1Ø IF A$>"5" THEN GOTO 5Ø
```

The next two statements are *NOT* coded correctly:

```
1Ø IF A = "N" THEN GOTO 5Ø
1Ø IF A$<1 THEN GOTO 5Ø
```

Now pause for a moment and complete the Programmer's Check which follows. Check your answers and be sure that you understand the material before continuing.

PROGRAMMER'S CHECK

1

Comparisons

1. In a numeric comparison:

(A) The values are made equal in length after decimal alignment and padding with zeros, if necessary.

(B) The values are made equal in length by padding the shorter field with blanks.

(C) The fields are compared byte-by-byte from left to right.

(D) Both B and C.

2. In an alphanumeric comparison:

(A) The values are made equal in length after decimal alignment and padding with zeros, if necessary.

(B) The values are made equal in length by padding the shorter field with blanks.

(C) The fields are compared byte-by-byte from left to right.

(D) Both B and C.

(continued)

Programmer's Check 1 (continued)

3. In the following examples, pick out the *line* number that will be executed *after* the IF statement (line 3Ø):

3A.

```
1Ø LET A$ = 'JOHN'

2Ø LET B$ = 'JOHNS'

3Ø IF A$>B$ THEN GOTO 5Ø

4Ø GOTO 1Ø

5Ø STOP
```

(A) 1Ø

(B) 3Ø

(C) 4Ø

(D) 5Ø

3B.

```
1Ø LET A = 3.1

2Ø LET B = Ø3.1Ø

3Ø IF A = B THEN GOTO 5Ø

4Ø GOTO 1Ø

5Ø STOP
```

(A) 1Ø

(B) 3Ø

(C) 4Ø

(D) 5Ø

3C.

```
1Ø LET A$ = "3.1"

2Ø LET B$ = "Ø3.1Ø"

3Ø IF A$ = B$ THEN GOTO 1Ø

4Ø STOP
```

(A) 1Ø

(B) 2Ø

(C) 3Ø

(D) 4Ø

4. Which of the following symbols represents the "not equal to" comparison?

(A) >

(B) <

(C) ><

(D) =

5. Why are these statements illogical?

```
1Ø IF A>B THEN GOTO 2Ø

2Ø STOP
```

(A) They are invalid statements in BASIC.

(B) Line 2Ø will be executed, regardless of the results of the comparison.

(C) Line 2Ø will never be executed.

(D) The statements *are* logical.

(Answers on Page 10)

PROGRAMMER'S CHECK ANSWERS

1

1. (A) The values are made equal in length after decimal alignment and padding with zeros, if necessary.

2. (D) Both B and C.

3.

   3A — (C) 4Ø

   3B — (D) 5Ø

   3C — (D) 4Ø

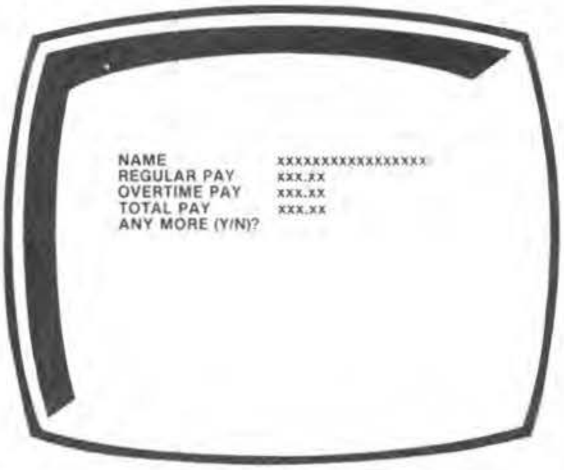
4. (C)

5. (B) Line 2Ø will be executed, regardless of the results of the comparison.

CODING A PAYROLL PROGRAM

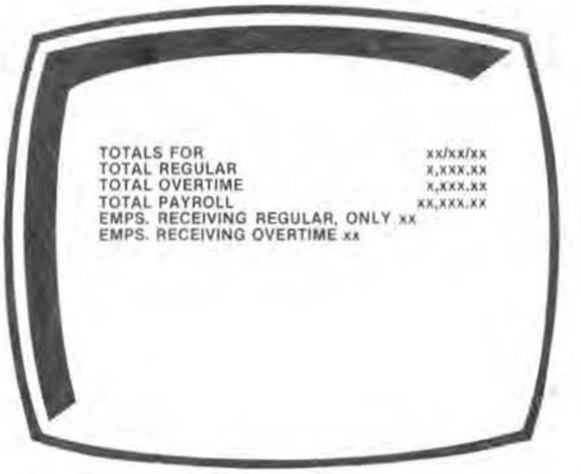
Now let's include the logic of comparing in a BASIC program. In this program, a payroll report will be displayed on the CRT. The output is illustrated below:

SCREEN 1



```
NAME            xxxxxxxxxxxxxxxxx
REGULAR PAY     xxx.xx
OVERTIME PAY    xxx.xx
TOTAL PAY       xxx.xx
ANY MORE (Y/N)?
```

SCREEN 2



```
TOTALS FOR                              xx/xx/xx
TOTAL REGULAR                           x,xxx.xx
TOTAL OVERTIME                          x,xxx.xx
TOTAL PAYROLL                          xx,xxx.xx
EMPS. RECEIVING REGULAR, ONLY xx
EMPS. RECEIVING OVERTIME xx
```

The input records for this program look like this:

INPUT

NAME	HOURS	RATE

CALCULATIONS

In this example, overtime pay is granted to those who have worked more than 40 hours a week. Overtime pay and regular pay are computed according to the following calculations:

(A) For those not working over 4Ø hours,

REGULAR PAY = HOURS * RATE

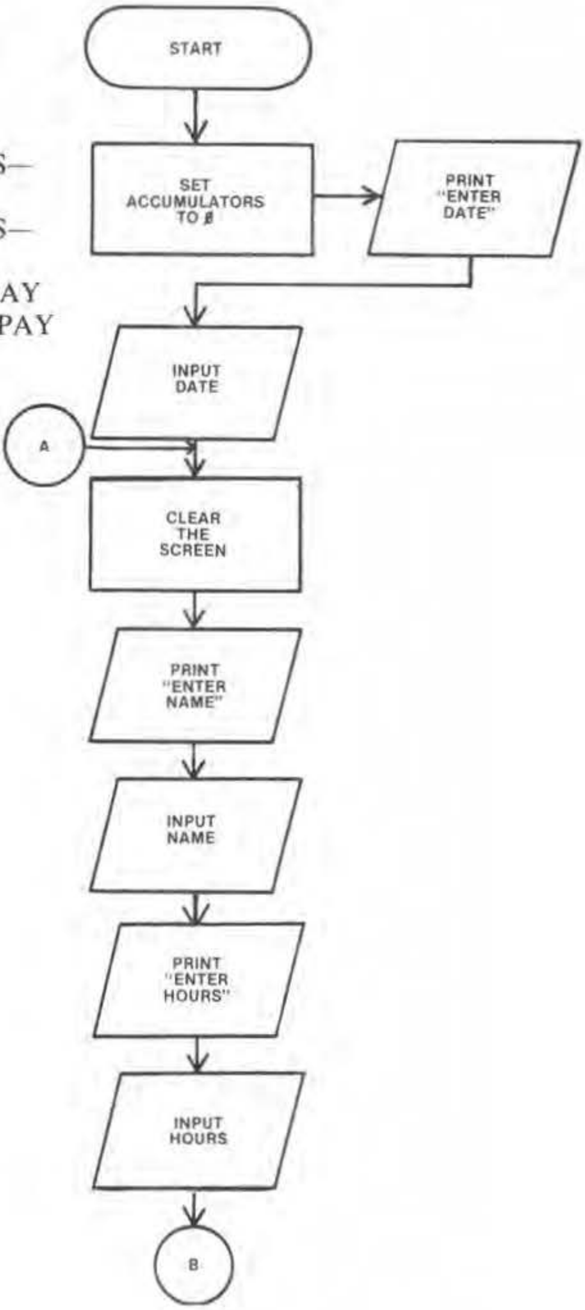
(B) For those working over 4Ø hours,

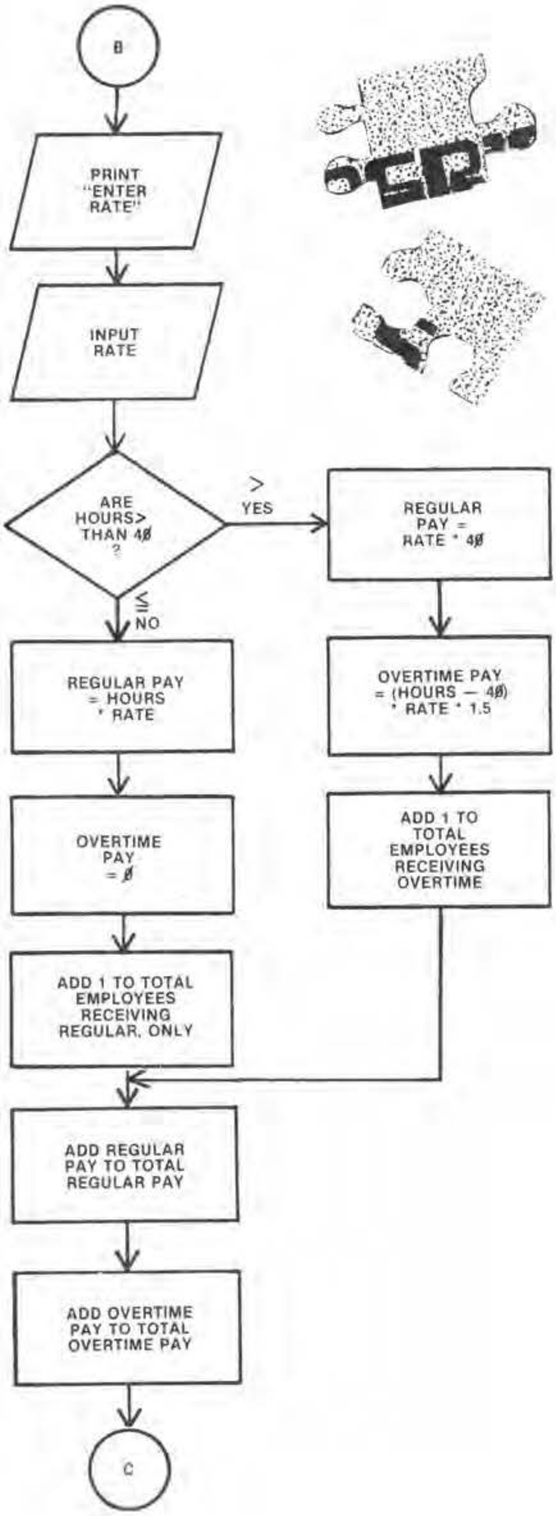
REGULAR PAY = 4Ø * RATE

OVERTIME PAY = (HOURS - 4Ø) * RATE * 1.5

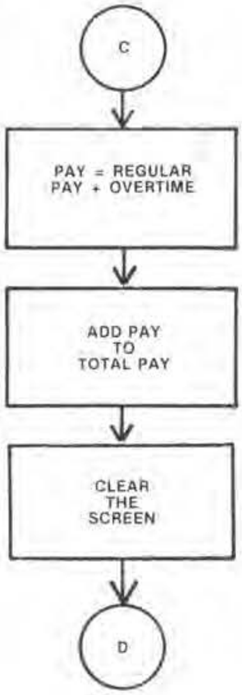
The flowchart on the following page designs the logic which will be used in coding this program.

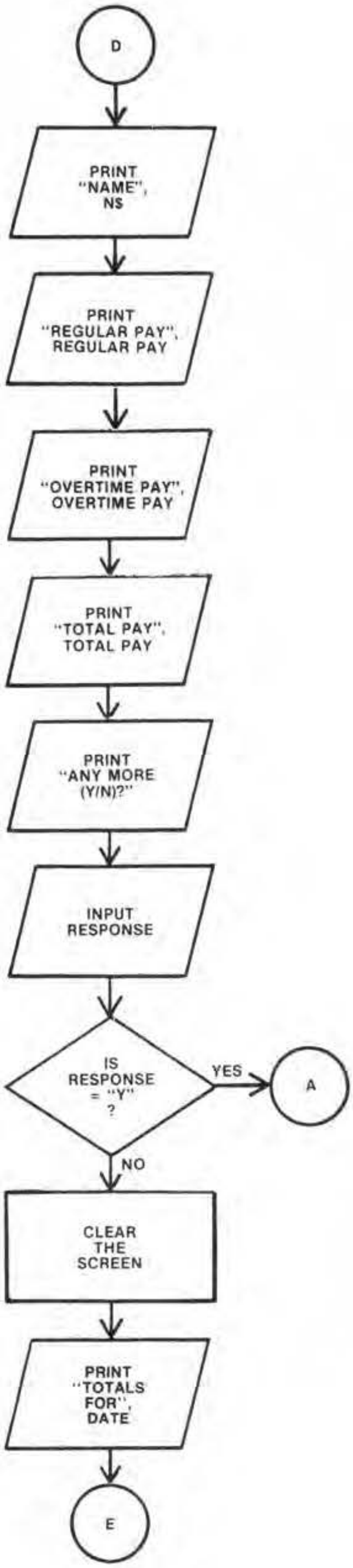
```
1Ø  REM IU7S1 (INSTRUCTION UNIT 7, SAMPLE 1)

2Ø  REM YOUR NAME

3Ø  REM DELETE THESE LINES IF
    STORAGE BECOMES FULL

 4Ø REM  VARIABLES      MEANINGS
 5Ø REM  D$             DATE — MM/DD/YY FORMAT
 6Ø REM  N$             EMPLOYEE'S NAME
 7Ø REM  H              HOURS WORKED
 8Ø REM  R              RATE OF PAY
1ØØ REM  A              REGULAR PAY
11Ø REM  B              OVERTIME PAY
13Ø REM  D              EMPLOYEE'S PAY
14Ø REM  T1             TOTAL EMPLOYEES—
                        REGULAR, ONLY
15Ø REM  T2             TOTAL EMPLOYEES—
                        OVERTIME
16Ø REM  T3             TOTAL REGULAR PAY
17Ø REM  T4             TOTAL OVERTIME PAY
19Ø REM  T6             TOTAL PAYROLL
2ØØ REM  M$             RESPONSE TO
                        CONTINUE LOOP


21Ø  LET T1 = Ø

22Ø  LET T2 = Ø

23Ø  LET T3 = Ø

24Ø  LET T4 = Ø

26Ø  LET T6 = Ø

27Ø  PRINT "ENTER DATE"

28Ø  INPUT D$

29Ø  CLS

3ØØ  PRINT "ENTER NAME"

31Ø  INPUT N$

32Ø  PRINT "ENTER HOURS"

33Ø  INPUT H
```



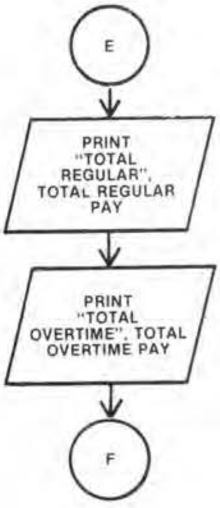

B
PRINT "ENTER RATE"
INPUT RATE
ARE HOURS> THAN 4Ø ?
>
YES
≦
NO
REGULAR PAY = RATE * 4Ø
OVERTIME PAY = (HOURS — 4Ø) * RATE * 1.5
ADD 1 TO TOTAL EMPLOYEES RECEIVING OVERTIME
REGULAR PAY = HOURS * RATE
OVERTIME PAY = Ø
ADD 1 TO TOTAL EMPLOYEES RECEIVING REGULAR, ONLY
ADD REGULAR PAY TO TOTAL REGULAR PAY
ADD OVERTIME PAY TO TOTAL OVERTIME PAY
C

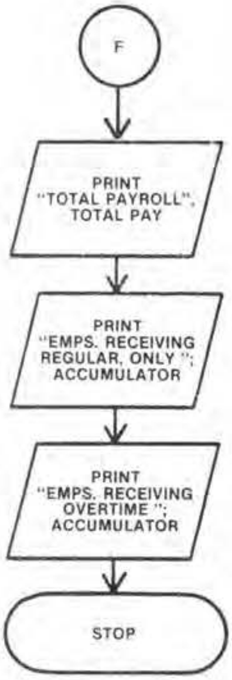
```
34Ø  PRINT "ENTER RATE"
35Ø  INPUT R
38Ø  IF H>4Ø THEN GOTO 43Ø
39Ø  LET A = H * R
4ØØ  LET B = Ø
41Ø  LET T1 = T1 + 1
42Ø  GOTO 46Ø
43Ø  LET A = R * 4Ø
44Ø  LET B = (H - 4Ø) * R * 1.5
45Ø  LET T2 = T2 + 1
46Ø  LET T3 = T3 + A
47Ø  LET T4 = T4 + B
52Ø  LET D = A + B
53Ø  LET T6 = T6 + D
54Ø  CLS
```





```
55Ø  PRINT "NAME", N$
56Ø  PRINT "REGULAR PAY", A
57Ø  PRINT "OVERTIME PAY", B
59Ø  PRINT "TOTAL PAY", D
6ØØ  PRINT "ANY MORE (Y/N)?"
61Ø  INPUT M$
62Ø  IF M$ = "Y" THEN GOTO 29Ø
63Ø  CLS
64Ø  PRINT "TOTALS FOR", D$
65Ø  PRINT "TOTAL REGULAR", T3
66Ø  PRINT "TOTAL OVERTIME", T4
68Ø  PRINT "TOTAL PAYROLL", T6
69Ø  PRINT "EMPS. RECEIVING
     REGULAR, ONLY "; T1
7ØØ  PRINT "EMPS. RECEIVING
     OVERTIME "; T2
71Ø  STOP
```


E
PRINT "TOTAL REGULAR", TOTAL REGULAR PAY
PRINT "TOTAL OVERTIME", TOTAL OVERTIME PAY
F


F
PRINT "TOTAL PAYROLL", TOTAL PAY
PRINT "EMPS. RECEIVING REGULAR, ONLY "; ACCUMULATOR
PRINT "EMPS. RECEIVING OVERTIME "; ACCUMULATOR
STOP

WALKTHROUGH OF SAMPLE PROGRAM

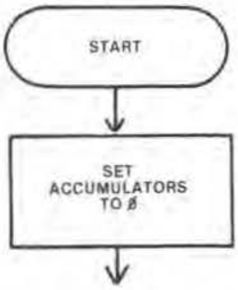
A step-by-step walkthrough with several records demonstrates how the compare logic controls the processing that takes place.

TEST INPUT RECORDS

DATE = Ø1/Ø8/83

NAME	HOURS	RATE
SMITH	45	6.ØØ
JONES	32	5.5Ø
DOE	42	5.ØØ

FLOWCHART

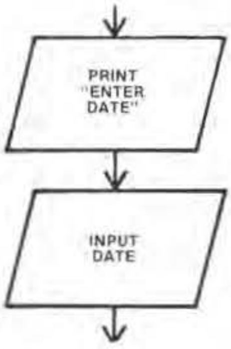


PROGRAM

```
21Ø  LET T1 = Ø
22Ø  LET T2 = Ø
23Ø  LET T3 = Ø
24Ø  LET T4 = Ø
26Ø  LET T6 = Ø
```

STEP 1

The initialization routine sets the six accumulators in this program to zero.

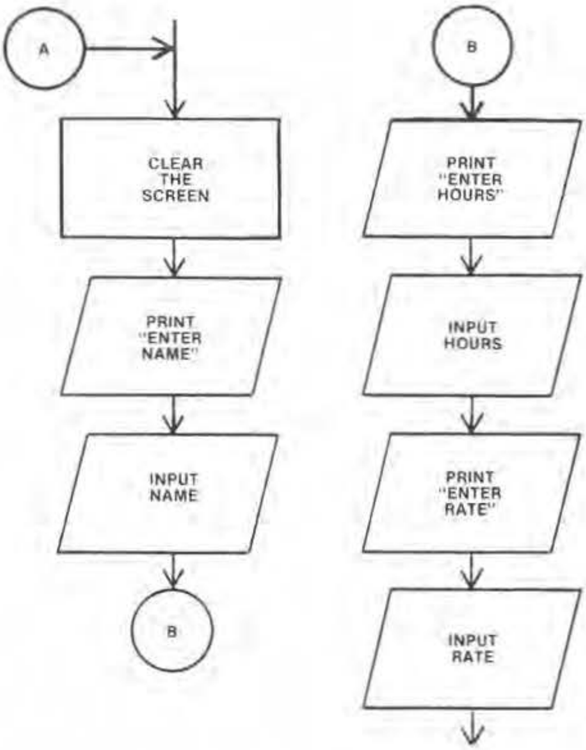


```
27Ø  PRINT
       "ENTER DATE"
28Ø  INPUT D$
```

STEP 2

A prompt is issued with a message preceding it. A value will be entered for D$ in the MM/DD/YY format (e.g. Ø1/Ø8/83). This date will be displayed later along with the final totals.

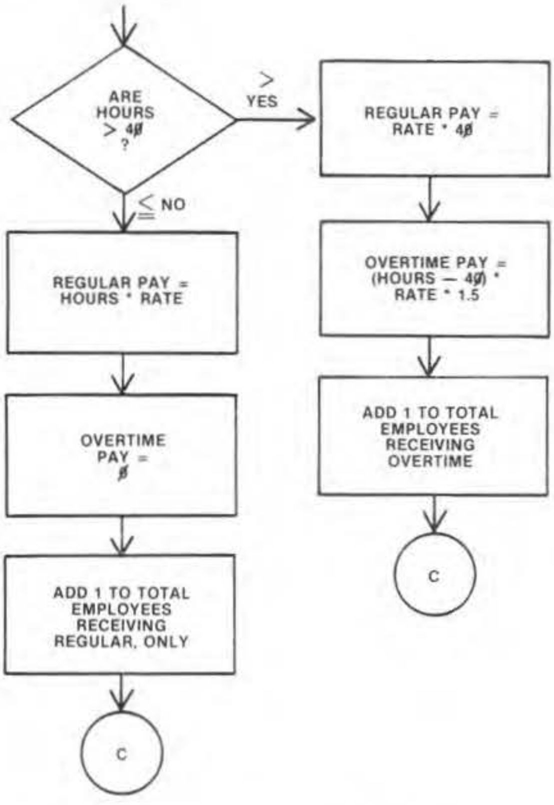
STEP 3



```
29Ø  CLS
3ØØ  PRINT "ENTER NAME"
31Ø  INPUT N$
32Ø  PRINT "ENTER HOURS"
33Ø  INPUT H
34Ø  PRINT "ENTER RATE"
35Ø  INPUT R
```

At this point, the screen is cleared and four prompts are issued for the input fields: name (N$), hours (H) and rate (R). We now would enter the values of SMITH, 45, and 6.ØØ respectively.

STEP 4


ARE HOURS > 4Ø ?
> YES
≤ NO
REGULAR PAY = RATE * 4Ø
OVERTIME PAY = (HOURS — 4Ø) * RATE * 1.5
ADD 1 TO TOTAL EMPLOYEES RECEIVING OVERTIME
C
REGULAR PAY = HOURS * RATE
OVERTIME PAY = Ø
ADD 1 TO TOTAL EMPLOYEES RECEIVING REGULAR, ONLY
C

```
38Ø  IF H>4Ø THEN GOTO 43Ø

39Ø  LET A = H * R

4ØØ  LET B = Ø

41Ø  LET T1 = T1 + 1

42Ø  GOTO 46Ø

43Ø  LET A = R * 4Ø

44Ø  LET B = (H - 4Ø) * R * 1.5

45Ø  LET T2 = T2 + 1
```

The first branch in our logic occurs with the IF statement at line 38Ø. Here, the hours (H) submitted as input are numerically compared against the constant, 4Ø. If the hours are greater than 4Ø, a branch to line 43Ø will occur; otherwise, line 39Ø will be executed. In the record we are processing, the value 45 was given as the hours, so the branch is performed.

At line 43Ø, the regular pay of the employee is calculated as rate times 4Ø. The rate given as input was 6.ØØ per hour, so the result of 24Ø.ØØ is stored under the variable name, A. Contrast this line with the equation for regular pay if hours are less than or equal to 4Ø (39Ø LET A = H * R). For those employees receiving overtime pay, we calculate their regular pay by multiplying their regular rate of pay by the first 4Ø hours they worked.

In the next statement, line 44Ø, the overtime pay is computed by first subtracting 4Ø from the hours worked (in order to find out how many overtime hours were worked) and then multiplying the overtime hours by the rate times 1.5. Overtime workers here get time-and-a-half (1.5) their normal rate of pay.

FIGURE 2—Executives must figure out such puzzles as projected sales, production schedules, and production costs. A salary and bonus pay program such as the one presented in your Study Unit can be used for planning purposes as well as actual week-to-week utility.

In this case:

(A) B = (H - 4Ø) * R * 1.5

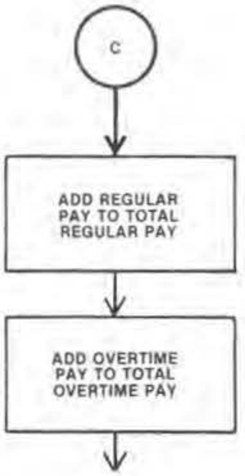
(B) B = (45 - 4Ø) * 6.ØØ * 1.5

(C) B = 5 * 6.ØØ * 1.5

(D) B = 45.ØØ

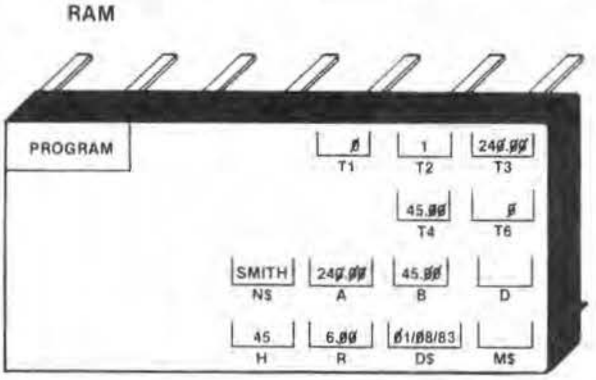
And, finally, since we are in a portion of the program at which only overtime employee records will be accessed, one is added to the accumulator, T2.

STEP 5

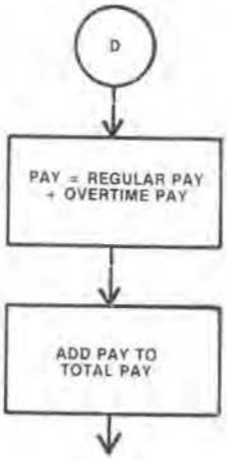

C
ADD REGULAR PAY TO TOTAL REGULAR PAY
ADD OVERTIME PAY TO TOTAL OVERTIME PAY

```
46Ø  LET T3 = T3 + A

47Ø  LET T4 = T4 + B
```

Regardless of the branching that may have taken place, the accumulators for the total regular pay and total overtime pay are added to. Internally, our RAM looks something like this:



STEP 6

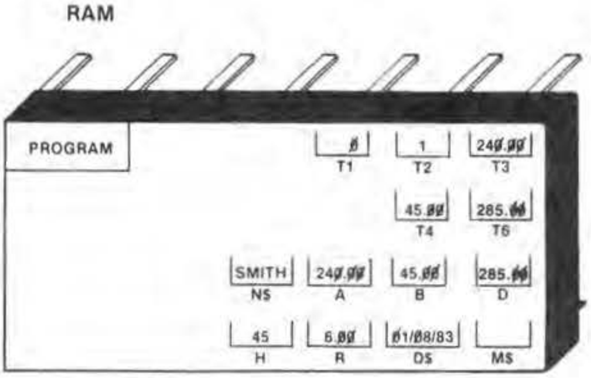


```
52Ø  LET D = A + B

53Ø  LET T6 = T6 + D
```

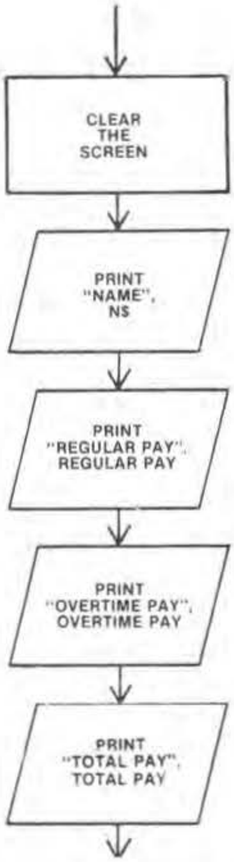
Next, the pay for an individual employee (D) is calculated as regular pay (A) plus overtime pay (B).

Lastly, this total is added to the total payroll accumulator (T6). Note how these values are used regardless of the different steps that have been executed previously.

Our record will calculate 285.ØØ for the pay and add this to T6. Main storage now looks like this:

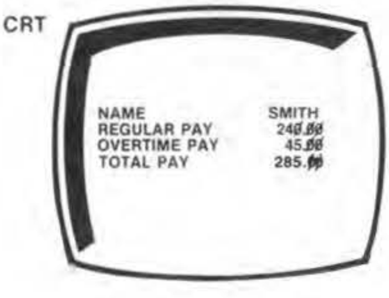


STEP 7


CLEAR THE SCREEN
PRINT "NAME", N$
PRINT "REGULAR PAY", REGULAR PAY
PRINT "OVERTIME PAY", OVERTIME PAY
PRINT "TOTAL PAY", TOTAL PAY

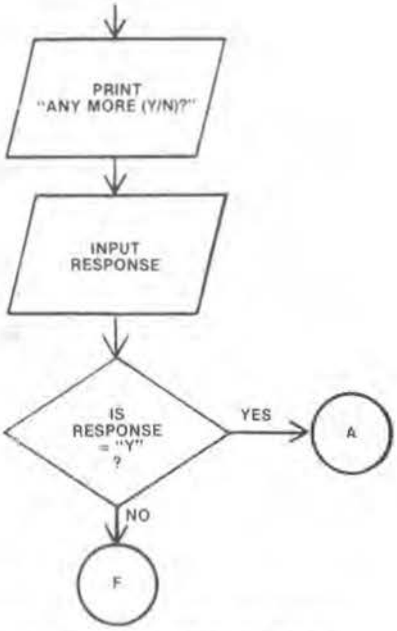
```
54Ø  CLS

55Ø  PRINT "NAME", N$

56Ø  PRINT "REGULAR PAY", A

57Ø  PRINT "OVERTIME PAY", B

59Ø  PRINT "TOTAL PAY ", D
```

This series of instructions displays on a cleared screen the results of our calculations and comparisons.



```
NAME            SMITH
REGULAR PAY     24Ø.ØØ
OVERTIME PAY     45.ØØ
TOTAL PAY       285.ØØ
```

STEP 8



```
6ØØ  PRINT "ANY MORE (Y/N)?"

61Ø  INPUT M$

62Ø  IF M$ = "Y" THEN GOTO 29Ø
```

Near the end of our loop, we encounter statements which will control the loop.

A message and a prompt appear, requesting the operator to enter either "Y" (yes) or "N" (no) as an answer to the question, "ANY MORE"?

A response of "Y" (for the value M$) causes a branch back to the instructions to clear the screen and input new data. A response of "N" (or for that matter, any other character entered) will cause the final totals to be displayed.

We would respond with "Y" and input the next record:

JONES, 32, 5.5Ø

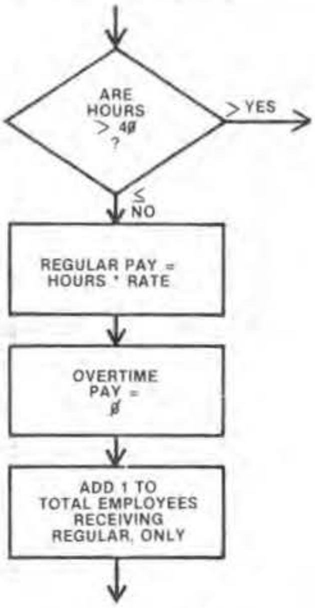
This time, when the IF statement at line 38Ø is encountered, no branching will occur, as Jones' hours (32) are *less than* 4Ø. Therefore, the regular pay (H) is determined by multiplying all the hours worked by the regular rate of pay (R).

OR

A = 32 * 5.5Ø

A = $176.ØØ

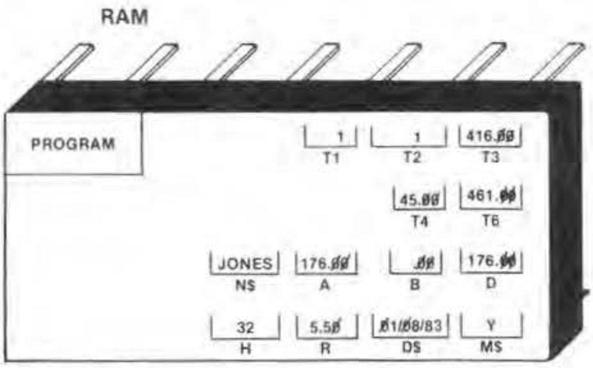
In the next statement, the overtime earnings are set back to Ø. If we were to have omitted this statement, B would still equal 45.ØØ, the value calculated for the *previous* record.


ARE HOURS > 4Ø ?
YES
≤ NO
REGULAR PAY = HOURS * RATE
OVERTIME PAY = Ø
ADD 1 TO TOTAL EMPLOYEES RECEIVING REGULAR, ONLY

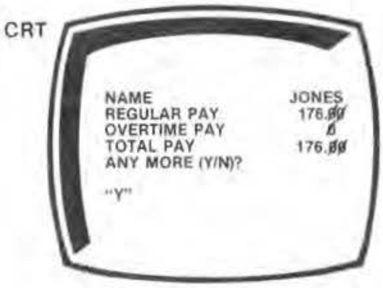
```
38Ø  IF H>4Ø THEN GOTO 43Ø

39Ø  LET A = H * R

4ØØ  LET B = Ø

41Ø  LET T1 = T1 + 1
```

As this record is processed, the appropriate accumulator will be incremented by 1 (T1) and the shared logic will be executed.

Main storage will appear as below, just before we are ready to enter data for the third record:



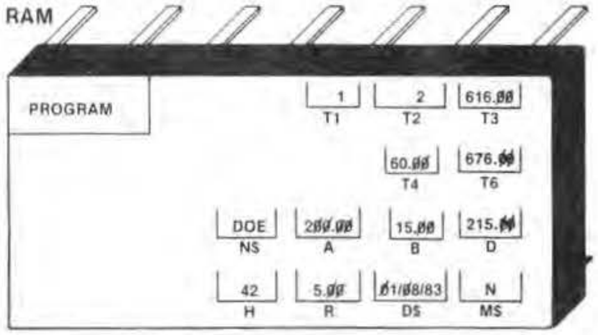
And our screen looks like this:

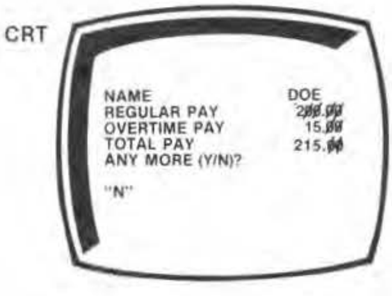


```
NAME                 JONES
REGULAR PAY         176.ØØ
OVERTIME PAY             Ø
TOTAL PAY           176.ØØ
ANY MORE (Y/N)?

"Y"
```

See if you can follow the path that the third record will take: (DOE, 42, 5.ØØ). Which set of instructions will be executed for calculating the regular and overtime pay?

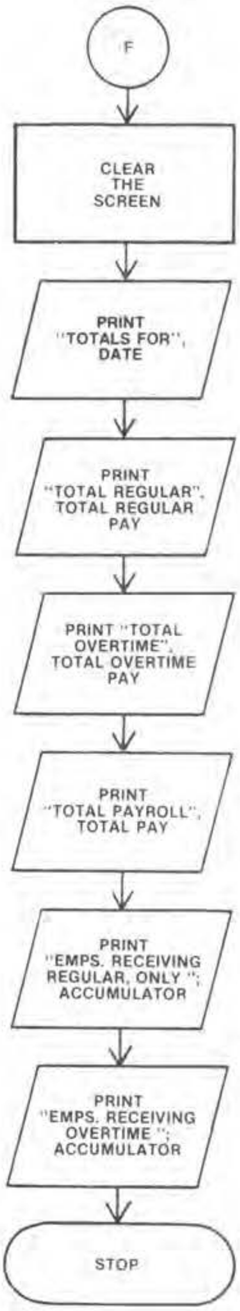
Here is what the RAM and the CRT will contain, just before the final totals are displayed (as we will respond with an "N" to the prompt on line 61Ø):





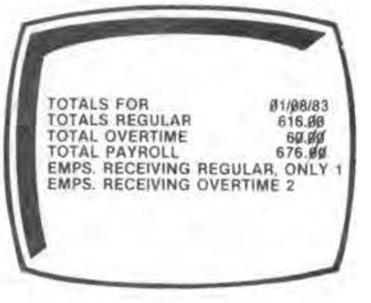
```
NAME                   DOE
REGULAR PAY         2ØØ.ØØ
OVERTIME PAY         15.ØØ
TOTAL PAY           215.ØØ
ANY MORE (Y/N)?

"N"
```

Our response of "N" to the prompt, "ANY MORE?," will have the logic "fall" into our final totals display screen. The values stored in RAM will be printed as follows:


F
CLEAR THE SCREEN
PRINT "TOTALS FOR", DATE
PRINT "TOTAL REGULAR", TOTAL REGULAR PAY
PRINT "TOTAL OVERTIME", TOTAL OVERTIME PAY
PRINT "TOTAL PAYROLL", TOTAL PAY
PRINT "EMPS. RECEIVING REGULAR, ONLY "; ACCUMULATOR
PRINT "EMPS. RECEIVING OVERTIME "; ACCUMULATOR
STOP

FINAL TOTAL ROUTINE

```
63Ø  CLS

64Ø  PRINT "TOTALS FOR", D$

65Ø  PRINT "TOTAL REGULAR", T3

66Ø  PRINT "TOTAL OVERTIME", T4

68Ø  PRINT "TOTAL PAYROLL", T6

69Ø  PRINT "EMPS. RECEIVING REGULAR,
     ONLY "; T1

7ØØ  PRINT "EMPS. RECEIVING
     OVERTIME "; T2

71Ø  STOP
```



```
TOTALS FOR                Ø1/Ø8/83
TOTALS REGULAR              616.ØØ
TOTAL OVERTIME               6Ø.ØØ
TOTAL PAYROLL               676.ØØ
EMPS. RECEIVING REGULAR, ONLY 1
EMPS. RECEIVING OVERTIME 2
```

This program can be expanded to include the logic for bonus pay. Before clearing memory you may wish to review the Appendix.

When you have fully understood this sample program, design and code the application problem in the following Programmer's Check.

PROGRAMMER'S CHECK

2

A Sales Discount Program

Program Name: IU7A1 (Instruction Unit 7, Assignment 1)

Type: SALES DISCOUNTS

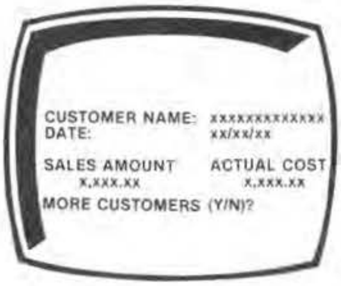
Specifications:

Companies often give discounts to customers who make large purchases. In this program, you are to give a 15% discount to all customers who purchase a total of $5øø.øø or less. A 2ø% discount is to be given for sales over $5øø.øø. For example, $3øø.øø in purchases would result in a discount of $45.øø ($3øø.øø * .15), but a sale of $75ø.øø gets a discount of $125.øø ($5øø.øø * .15 + (75ø.øø - 5øø.øø) * .2ø).

This program is to display one output screen for each customer, listing the customer's name, the date of the purchase, and the actual cost (sales amount minus discount).

OUTPUT

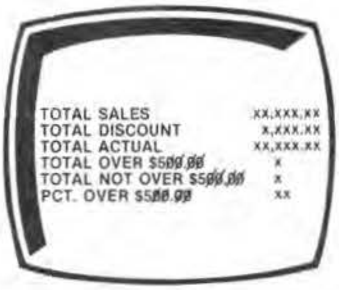
EXAMPLE: "DETAIL" SCREEN



```
CUSTOMER NAME:  xxxxxxxxxxxxxx
DATE:           xx/xx/xx

SALES AMOUNT     ACTUAL COST
   x,xxx.xx        x,xxx.xx
MORE CUSTOMERS (Y/N)?
```

After all records have been processed, the final screen should appear as below:

EXAMPLE: "FINAL" SCREEN



```
TOTAL SALES               xx,xxx.xx
TOTAL DISCOUNT             x,xxx.xx
TOTAL ACTUAL              xx,xxx.xx
TOTAL OVER $5øø.øø            x
TOTAL NOT OVER $5øø.øø        x
PCT. OVER $5øø.øø             xx
```

The input records will contain data for the date of purchase, the customer's name, the sales amount.

EXAMPLE: INPUT RECORD LAYOUT

DATE (MM/DD/YY)	CUSTOMER'S NAME	SALES AMOUNT

Programmer's Check 2 (continued)

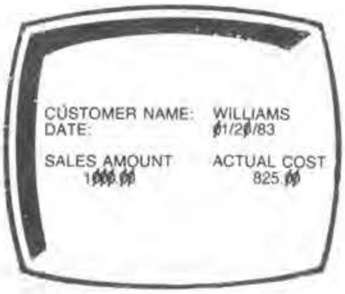
SAMPLE PROCESSING

Before beginning the designing, coding and testing of the program, make sure you understand what is required. For example, if we were to process a record like this:

DATE	NAME	AMOUNT
Ø1/2Ø/83	WILLIAMS	$1ØØØ.ØØ

the following output would be displayed:

EXAMPLE: CRT



```
CUSTOMER NAME:  WILLIAMS
DATE:           Ø1/2Ø/83

SALES AMOUNT    ACTUAL COST
   1ØØØ.ØØ         825.ØØ
```

The actual cost was determined by following these steps:

1. **This amount being over $5ØØ.ØØ, a sales discount of $175.ØØ was computed.**

```
SALES     = 5ØØ.ØØ * .15 +
DISCOUNT  (1ØØØ.ØØ - 5ØØ.ØØ) * .2Ø

          = 75.ØØ + 5ØØ.ØØ * .2Ø

          = 75.ØØ + 1ØØ.ØØ

          = $175.ØØ
```

2. **When this discount is then subtracted from $1,ØØØ.ØØ, the actual cost to the customer will be $825.ØØ**

3. **The number of customers over $5ØØ.ØØ in sales accumulator should be incremented. The percentage (PCT.) of sales over $5ØØ.ØØ on the final screen is found by dividing the number of sales over $5ØØ.ØØ by the total number of sales, and then multiplying by 1ØØ.**

When you have designed and coded the program, test it with the following input data:

DATE	NAME	AMOUNT
Ø1/2Ø/83	WILLIAMS	1ØØØ.ØØ
Ø1/21/83	ADAMS	45Ø.ØØ
Ø1/22/83	JOHNSON	6ØØ.ØØ
Ø1/23/83	THOMAS	5ØØ.ØØ

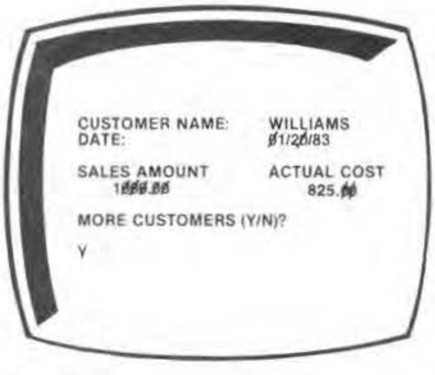
(Answers on Pages 22-25)

PROGRAMMER'S CHECK ANSWERS

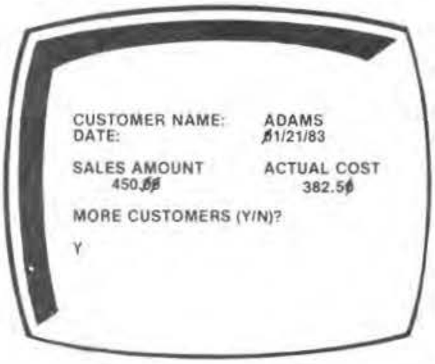
2

Your output should appear as:

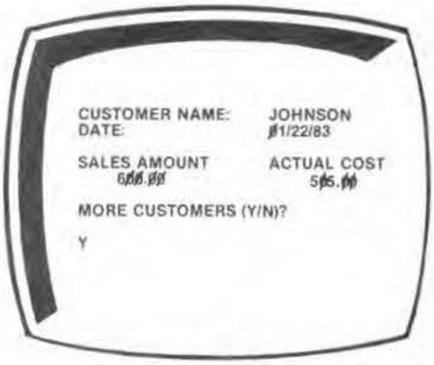
OUTPUT



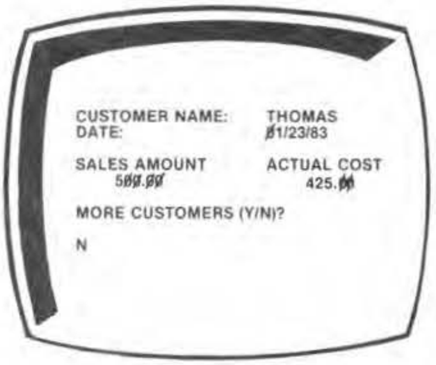
```
CUSTOMER NAME:     WILLIAMS
DATE:              Ø1/2Ø/83

SALES AMOUNT       ACTUAL COST
    1ØØØ.ØØ             825.ØØ

MORE CUSTOMERS (Y/N)?

Y
```



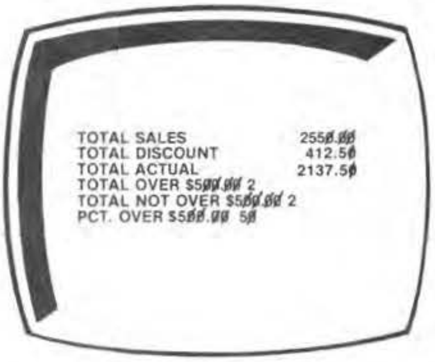
```
CUSTOMER NAME:     ADAMS
DATE:              Ø1/21/83

SALES AMOUNT       ACTUAL COST
    450.ØØ              382.5Ø

MORE CUSTOMERS (Y/N)?

Y
```



```
CUSTOMER NAME:     JOHNSON
DATE:              Ø1/22/83

SALES AMOUNT       ACTUAL COST
    6ØØ.ØØ              5Ø5.ØØ

MORE CUSTOMERS (Y/N)?

Y
```



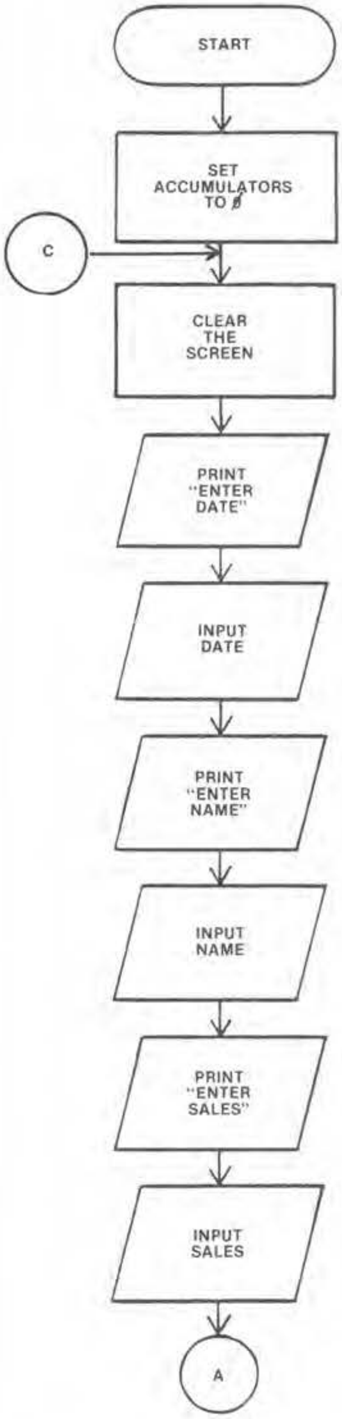
```
CUSTOMER NAME:     THOMAS
DATE:              Ø1/23/83

SALES AMOUNT       ACTUAL COST
    5ØØ.ØØ              425.ØØ

MORE CUSTOMERS (Y/N)?

N
```



```
TOTAL SALES                  255Ø.ØØ
TOTAL DISCOUNT                412.5Ø
TOTAL ACTUAL                 2137.5Ø
TOTAL OVER $5ØØ.ØØ 2
TOTAL NOT OVER $5ØØ.ØØ 2
PCT. OVER $5ØØ.ØØ  5Ø
```

Programmer's Check 2 Answer (continued)

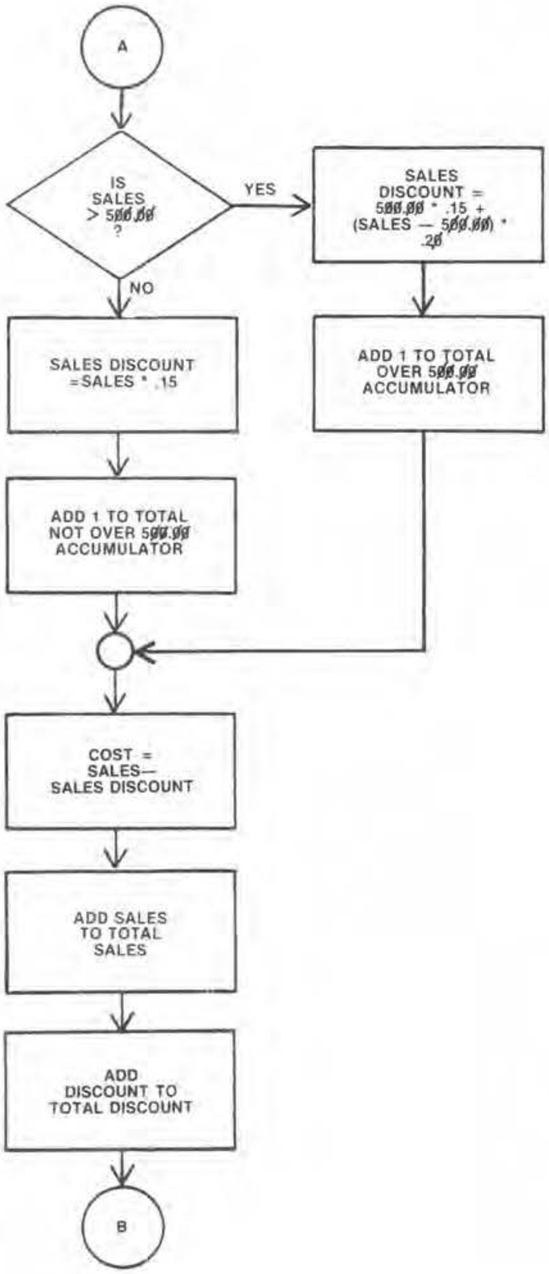
FLOWCHART SOLUTION TO IU7A1



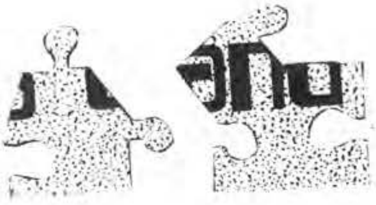
Program Solution to IU7A1

```
 1Ø REM  IU7A1       YOUR NAME
 2Ø REM  VARIABLES   MEANINGS
 3Ø REM  D$          DATE
 4Ø REM  N$          NAME
 5Ø REM  A           SALES AMOUNT
 8Ø REM  D2          SALES DISCOUNT
1ØØ REM  C           ACTUAL COST
11Ø REM  T1          TOTAL NOT OVER 5ØØ.ØØ
12Ø REM  T2          TOTAL OVER 5ØØ.ØØ
13Ø REM  T3          TOTAL SALES
14Ø REM  T4          TOTAL DISCOUNT
15Ø REM  T5          TOTAL ACTUAL COST
16Ø REM  P           PERCENTAGE OVER 5ØØ.ØØ
165 REM  R$          RESPONSE TO PROMPT (Y/N)

17Ø LET T1 = Ø

18Ø LET T2 = Ø

19Ø LET T3 = Ø

2ØØ LET T4 = Ø

21Ø LET T5 = Ø

22Ø CLS

23Ø PRINT "ENTER DATE"

24Ø INPUT D$

242 PRINT "ENTER NAME"

246 INPUT N$

25Ø PRINT "ENTER SALES"

26Ø INPUT A
```

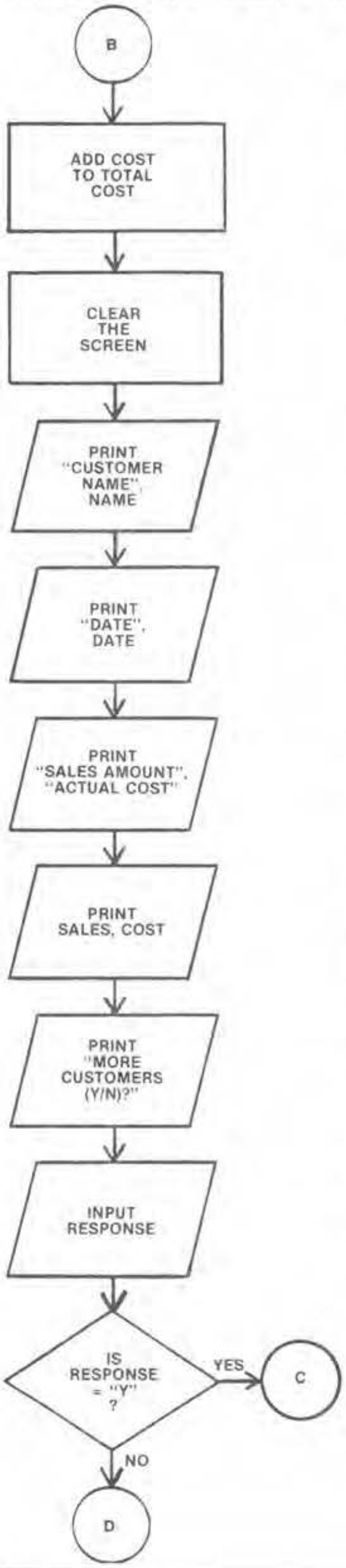
Programmer's Check 2 Answer (continued)


A
IS SALES > 5ØØ.ØØ ?
YES
NO
SALES DISCOUNT = 5ØØ.ØØ * .15 + (SALES — 5ØØ.ØØ) * .2Ø
SALES DISCOUNT =SALES * .15
ADD 1 TO TOTAL OVER 5ØØ.ØØ ACCUMULATOR
ADD 1 TO TOTAL NOT OVER 5ØØ.ØØ ACCUMULATOR
COST = SALES— SALES DISCOUNT
ADD SALES TO TOTAL SALES
ADD DISCOUNT TO TOTAL DISCOUNT
B

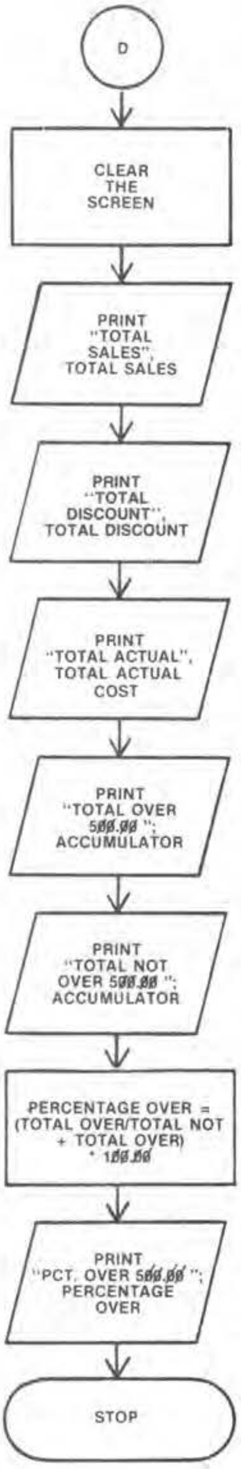
```
34Ø IF A>5ØØ.ØØ THEN GOTO 38Ø

35Ø LET D2 = A * .15

36Ø LET T1 = T1 + 1

37Ø GOTO 4ØØ

38Ø LET D2 = 75.ØØ + (A - 5ØØ.ØØ)
    * .2Ø

39Ø LET T2 = T2 + 1

4ØØ LET C = A - D2

41Ø LET T3 = T3 + A

42Ø LET T4 = T4 + D2
```



Programmer's Check 2 Answer (continued)



```
43Ø LET T5 = T5 + C

44Ø CLS

45Ø PRINT "CUSTOMER
    NAME",N$

46Ø PRINT "DATE",D$

47Ø PRINT "SALES AMOUNT",
    'ACTUAL COST"

48Ø PRINT TAB 4; A; TAB 19; C

49Ø PRINT "MORE CUSTOMERS
    (Y/N)?"

5ØØ INPUT R$

51Ø IF R$ = "Y" THEN GOTO 22Ø

52Ø CLS

53Ø PRINT "TOTAL SALES "; TAB
    22; T3

54Ø PRINT "TOTAL DISCOUNTS ";
    TAB 22; T4

55Ø PRINT "TOTAL ACTUAL ";
    TAB 22; T5

56Ø PRINT "TOTAL OVER 5ØØ.ØØ";
    TAB 22; T2

57Ø PRINT "TOTAL NOT OVER
    5ØØ.ØØ "; TAB 22l T1

58Ø LET P = (T2/(TI + T2)) *
    1ØØ.ØØ

59Ø PRINT "PCT. OVER 5ØØ.ØØ";
    TAB 22;P

6ØØ STOP
```


D
CLEAR THE SCREEN
PRINT "TOTAL SALES", TOTAL SALES
PRINT "TOTAL DISCOUNT", TOTAL DISCOUNT
PRINT "TOTAL ACTUAL", TOTAL ACTUAL COST
PRINT "TOTAL OVER 5ØØ.ØØ "; ACCUMULATOR
PRINT "TOTAL NOT OVER 5ØØ.ØØ "; ACCUMULATOR
PERCENTAGE OVER = (TOTAL OVER/TOTAL NOT + TOTAL OVER) * 1ØØ.ØØ
PRINT "PCT. OVER 5ØØ.ØØ "; PERCENTAGE OVER
STOP

COMPOUND COMPARISONS

The logical operators AND and OR can be used to make compound IF statements. Either one or both of them in a statement force the computer to check to see if more than one condition must be true in order for the entire statement to be true.

OR

The use of the word OR in an IF statement means that one *OR* the other conditions must be true for the THEN clause to be executed. For example, in the instructions:

```
1Ø INPUT A

2Ø INPUT B

3Ø IF A = 1 OR B>5 THEN GOTO 5Ø

4Ø GOTO 1Ø

5Ø STOP
```

FIGURE 3—Insurance underwriters can obtain instant summaries of auto insurance sold according to risk factor, driver age, and other variables—thanks to good programming.

Line 3Ø will branch to STOP if either a value of "1" is entered for A *OR* a value greater than "5" is entered for B. Otherwise, the program will continue to loop. Note that if the program stops, we wouldn't have any way of knowing *which* of the two conditions were true (or, if both were true, for that matter)!

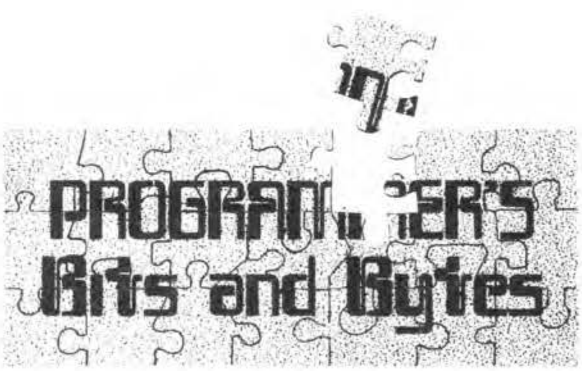
AND

The logical operator "AND" is employed when *both* conditions of either side of it must be true for the entire IF statement to be considered true.

Let's change our previous program to this:

```
1Ø INPUT A

2Ø INPUT B

3Ø IF A = 1 AND B>5 THEN GOTO 5Ø

4Ø GOTO 1Ø

5Ø STOP
```

In this case, the "AND" in line 3Ø will allow the program to branch to line 5Ø when *both* a value of "1" is entered for A *AND* a value greater than "5" is entered for B.



A SUMMARY OF CONDITIONAL STATEMENTS

Now, let's review what we have learned about conditional statements:

1. Conditional statements are coded by using the keyword "IF".

2. "IF" is followed by a conditional expression. This expression compares two values.

3. The result of the comparison can be true or false.

4. Six logical operators can be used:

   a. > (Greater than)
   b. < (Less than)
   c. = (Equal to)
   d. >= (Greater than or equal to)
   e. <= (Less than or equal to)
   f. <> (Less than or greater than—Not equal to)

5. If the result of the comparison is true, the statement which follows the word "THEN' will be executed.

6. If the result of the comparison is false, the next line in the program will be executed.

7. Numeric variables may be compared with other numeric variables or constants.

8. String variables may be compared with other string variables or constants.

9. When string constants are used in an IF statement, they *must* be enclosed within quotation (" ") marks.

10. Compound comparisons can be made using the words "AND" and "OR".

11. When "AND" is used to connect two conditions, both must be true for the whole comparison to be considered true.

12. When "OR" is used to connect two conditions, either one or both must be true for the whole comparison to be considered true.

13. A decision box is used when flowcharting comparisons. The conditional expression is stated as a question inside the box. The flowlines leading out of the box are labled as the "YES" and "NO" paths and are connected to the appropriate next instruction to be executed.

Now, it's time once again, to put your instruction into *action*. Complete Programmer's Check #3 which follows.

REPORT CODES

You are aware, now, that your computer will ask you for a keyword with a reverse K, or ask you for other characters (or INPUT when you are running a program) with a reverse L. It will advise you that you've made a syntax error, with a reverse S, and *refuses* to enter the line until you correct it.

When you are running a program, there are other signals that will be valuable to you. These *REPORT CODES* appear in the back of your USER MANUAL.

Example: Enter the following program:

```
1Ø LET X = 5
2Ø LET Y = X + N
3Ø PRINT Y
RUN
```

What do you get? The report 2/2Ø. Referring to your REPORT CODES, you will note that your computer told you that you have an *unassigned variable* (you did not assign a value to N); AND that the problem was on Line 2Ø.

Now, LIST your program and enter:

15 LET N = 1Ø

and your program will run.

You will find the REPORT CODES in the back of your USER MANUAL to be valuable, as you progress in programming.

PROGRAMMER'S CHECK

3

Insurance Rate Application Program

Program Name: IU7A2 (Instruction Unit 7, Assignment 2)

Type: COMPARING

Specifications:

EXAMPLE: INPUT

DRIVER'S NAME	CAR MODEL	AGE	RISK CODE

PROCESSING

Drivers are to be assigned car insurance rates, depending on two factors: their age and their risk code. The rates to be given are found in this chart:

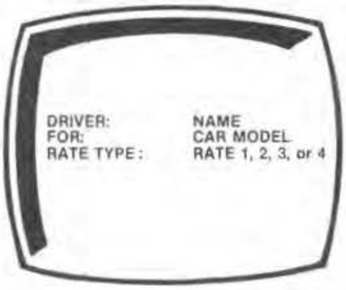
RATE CHART

TYPE	AGE	RISK CODE
RATE 1	OVER 25	A
RATE 2	OVER 25	B
RATE 3	NOT OVER 25	A
RATE 4	NOT OVER 25	B

OUTPUT

A screen should be displayed for each driver as shown below:

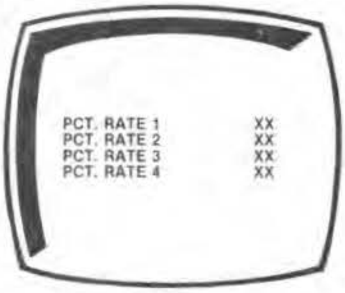
EXAMPLE: "DETAIL" SCREEN



```
DRIVER:       NAME
FOR:          CAR MODEL
RATE TYPE:    RATE 1, 2, 3, or 4
```

When all records have been entered, a screen displaying the percentage of each rate type should be printed:

EXAMPLE: "TOTAL" SCREEN



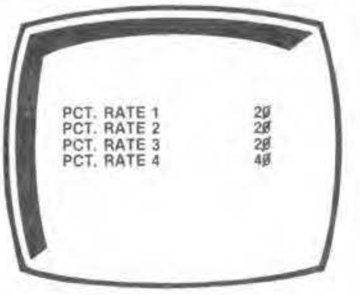
```
PCT. RATE 1        XX
PCT. RATE 2        XX
PCT. RATE 3        XX
PCT. RATE 4        XX
```

Use the following records to test your program.

NAME	CAR	AGE	RISK CODE
JONES	FORD	24	A
SMITH	CHEVY	3Ø	B
THOMPSON	DODGE	25	B
HUNT	BUICK	21	B
MILLER	PLYMOUTH	26	A

Programmer's Check 3 (continued)

EXAMPLE: "TOTAL" SCREEN



```
PCT. RATE 1          2Ø
PCT. RATE 2          2Ø
PCT. RATE 3          2Ø
PCT. RATE 4          4Ø
```

(Answers on Pages 30-33)

EXTRA ASSIGNMENTS

(A) Write a program which will convert Fahrenheit temperatures to Celsius or vice versa. The user should be prompted for a temperature and whether it is to be converted to Fahrenheit ("F") or Celsius ("C"). The calculations are:

Celsius = (Fahrenheit - 32) * (5/9)

Fahrenheit = Celsius * (9/5) + 32

(B) Write a program which incorporates many different metric conversions, such as:

Ounces to grams (1 to 28.4)

Pounds to kilograms (2.2 to 1)

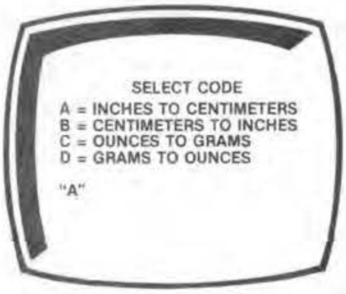
Miles to kilometers (1 to 1.6)

Inches to centimeters (1 to 2.54)

Display a screen which prints all of the available conversions and a code. Then prompt the user for the code and the value. The program will then branch to the calculation and print the output. For example:

MENU SCREEN

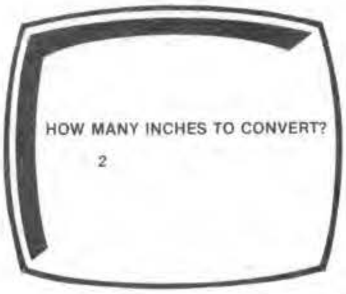
SCREEN 1



```
             SELECT CODE
A = INCHES TO CENTIMETERS
B = CENTIMETERS TO INCHES
C = OUNCES TO GRAMS
D = GRAMS TO OUNCES

"A"
```

If "A" is chosen, the following would be displayed:

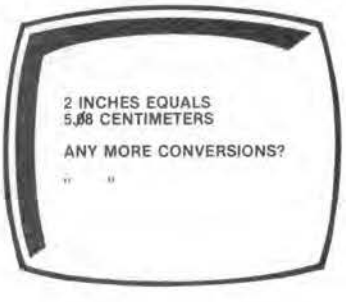
SCREEN 2



```
HOW MANY INCHES TO CONVERT?

      2
```

If 2 was entered, the output might appear as:

SCREEN 3

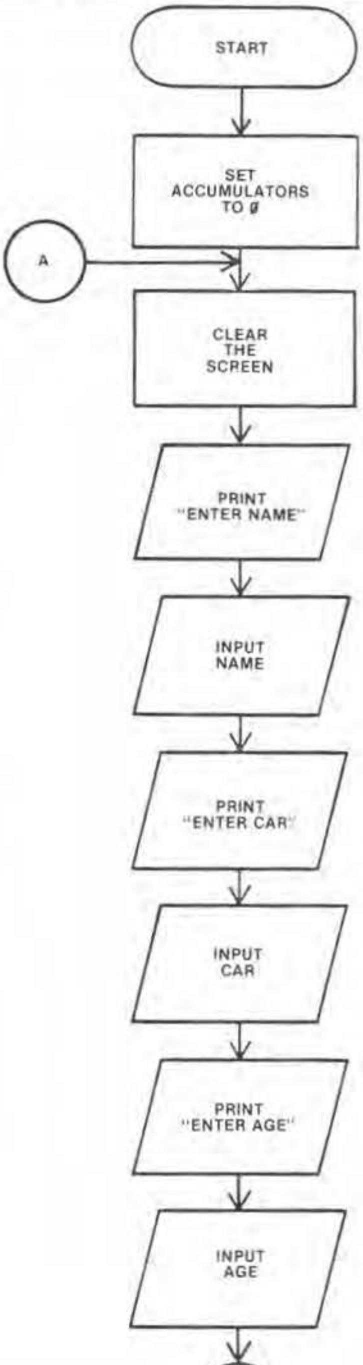


```
2 INCHES EQUALS
5.Ø8 CENTIMETERS

ANY MORE CONVERSIONS?

"    "
```

PROGRAMMER'S CHECK ANSWERS

3

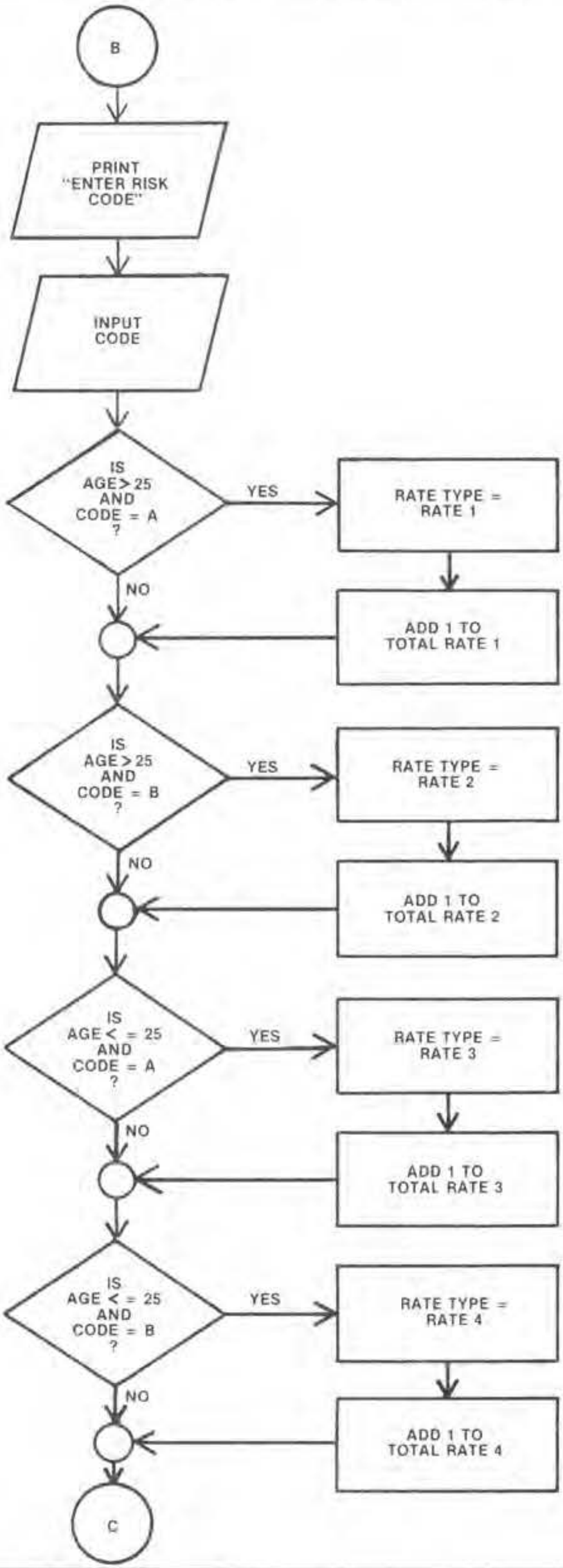
FLOWCHART SOLUTION TO IU7A2



Program Solution to IU7A2

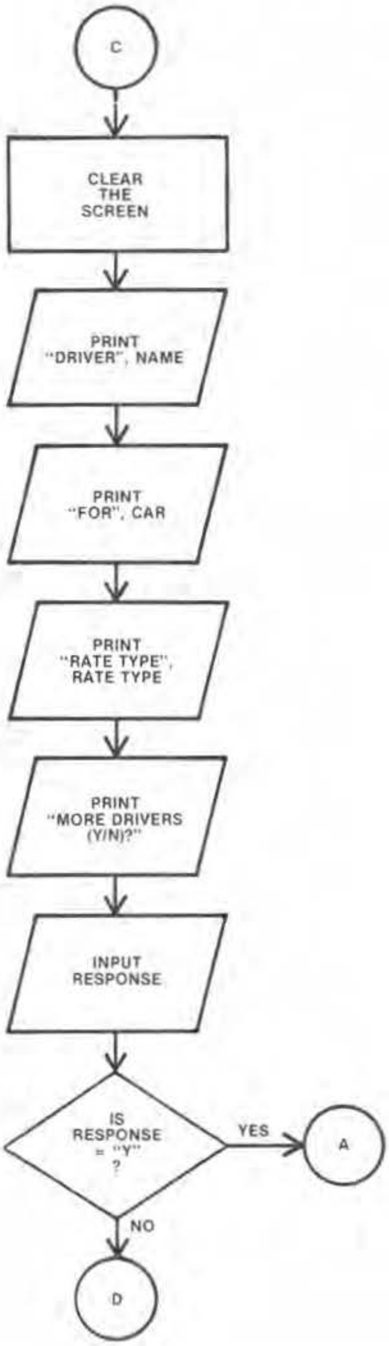
```
 1Ø REM  IU7A2          YOUR NAME
 2Ø REM  VARIABLES      MEANINGS
 3Ø REM  N$             NAME
 4Ø REM  C$             CAR TYPE
 5Ø REM  A              AGE OF DRIVER
 6Ø REM  R$             RISK CODE
 7Ø REM  T$             RATE TYPE
 8Ø REM  T1             TOTAL - RATE 1
 9Ø REM  T2             TOTAL - RATE 2
1ØØ REM  T3             TOTAL - RATE 3
11Ø REM  T4             TOTAL - RATE 4
12Ø REM  T5             TOTAL DRIVERS
13Ø REM  Q$             PROMPT RESPONSE

14Ø LET T1 = Ø

15Ø LET T2 = Ø

16Ø LET T3 = Ø

17Ø LET T4 = Ø

18Ø LET T5 = Ø

19Ø CLS

2ØØ PRINT "ENTER NAME"

21Ø INPUT N$

22Ø PRINT "ENTER CAR"

23Ø INPUT C$

24Ø PRINT "ENTER AGE"

25Ø INPUT A
```

Programmer's Check 3 Answer (continued)



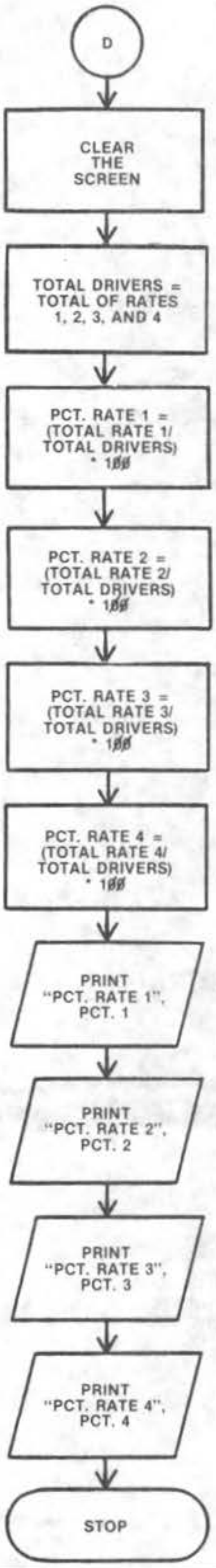
```
26Ø PRINT "ENTER RISK CODE"
27Ø INPUT R$
28Ø IF A>25 AND R$ = "A"
    THEN GOTO 32Ø
29Ø IF A>25 AND R$ = "B"
    THEN GOTO 35Ø
3ØØ IF A< = 25 AND R$ = "A"
    THEN GOTO 38Ø
31Ø IF A< = 25 AND R$ = "B"
    THEN GOTO 41Ø
32Ø LET T$ = "RATE 1"
33Ø LET T1 = T1 + 1
34Ø GOTO 43Ø
35Ø LET T$ = "RATE 2"
36Ø LET T2 = T2 + 1
37Ø GOTO 43Ø
38Ø LET T$ = "RATE 3"
39Ø LET T3 = T3 + 1
4ØØ GOTO 43Ø
41Ø LET T$ = "RATE 4"
42Ø LET T4 = T4 + 1
```

Programmer's Check 3 Answer (continued)

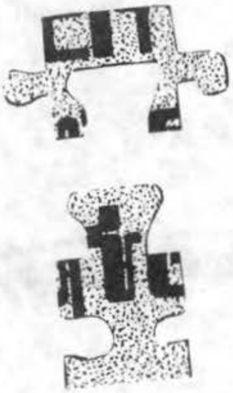


```
430 CLS
440 PRINT "DRIVER", N$
450 PRINT "FOR", C$
460 PRINT "RATE TYPE", T$
470 PRINT "MORE DRIVERS
    (Y/N)?"
480 INPUT Q$
490 IF Q$ = "Y" THEN GOTO 190
```

Programmer's Check 3 Answer (continued)


D
CLEAR THE SCREEN
TOTAL DRIVERS = TOTAL OF RATES 1, 2, 3, AND 4
PCT. RATE 1 = (TOTAL RATE 1/ TOTAL DRIVERS) * 1ØØ
PCT. RATE 2 = (TOTAL RATE 2/ TOTAL DRIVERS) * 1ØØ
PCT. RATE 3 = (TOTAL RATE 3/ TOTAL DRIVERS) * 1ØØ
PCT. RATE 4 = (TOTAL RATE 4/ TOTAL DRIVERS) * 1ØØ
PRINT "PCT. RATE 1", PCT. 1
PRINT "PCT. RATE 2", PCT. 2
PRINT "PCT. RATE 3", PCT. 3
PRINT "PCT. RATE 4", PCT. 4
STOP

```
5ØØ CLS

51Ø LET T5 = T1 + T2 + T3 + T4

52Ø PRINT "PCT. RATE 1",
    (T1/T5) * 1ØØ

53Ø PRINT "PCT. RATE 2",
    (T2/T5) * 1ØØ

54Ø PRINT "PCT. RATE 3",
    (T3/T5) * 1ØØ

55Ø PRINT "PCT. RATE 4",
    (T4/T5) * 1ØØ

56Ø STOP
```



APPENDIX

ADDITIONAL PROGRAMMING LINES FOR PAYROLL PROGRAM

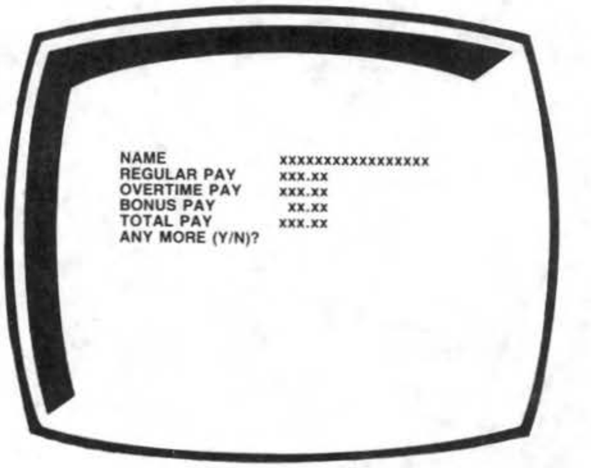
The sample payroll program could be modified to include bonus pay for employees based upon a bonus code entered as input according to the following table:

BONUS PAY CHART

BONUS CODE ON RECORD	BONUS PAY
A	$2Ø.ØØ
B	25.ØØ
C	3Ø.ØØ

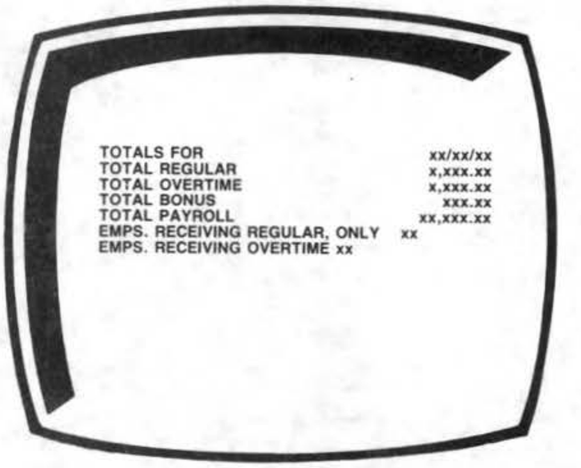
We will now produce screen output like this:

SCREEN 1



```
NAME               xxxxxxxxxxxxxxxxx
REGULAR PAY        xxx.xx
OVERTIME PAY       xxx.xx
BONUS PAY           xx.xx
TOTAL PAY          xxx.xx
ANY MORE (Y/N)?
```

SCREEN 2



```
TOTALS FOR                              xx/xx/xx
TOTAL REGULAR                           x,xxx.xx
TOTAL OVERTIME                          x,xxx.xx
TOTAL BONUS                               xxx.xx
TOTAL PAYROLL                          xx,xxx.xx
EMPS. RECEIVING REGULAR, ONLY   xx
EMPS. RECEIVING OVERTIME xx
```

The program could be changed as shown below. The lines that need to be added to your program have been marked with an asterisk (*), lines that need editing have been marked with two asterisks (**). You can easily add these lines to your program by listing your program and keyboarding these lines in.

```
  1Ø  REM IU7S1 (INSTRUCTION UNIT 7, SAMPLE 1)

  2Ø  REM YOUR NAME

  3Ø  REM DELETE THESE LINES IF
     STORAGE BECOMES FULL

  4Ø REM  VARIABLES       MEANINGS
  5Ø REM  D$              DATE — MM/DD/YY FORMAT
  6Ø REM  N$              EMPLOYEE'S NAME
  7Ø REM  H               HOURS WORKED
  8Ø REM  R               RATE OF PAY
 *9Ø REM  B$              BONUS CODE
 1ØØ REM  A               REGULAR PAY
 11Ø REM  B               OVERTIME PAY
*12Ø REM  C               BONUS PAY
 13Ø REM  D               EMPLOYEE'S PAY
 14Ø REM  T1              TOTAL EMPLOYEES—
                          REGULAR, ONLY
 15Ø REM  T2              TOTAL EMPLOYEES—
                          OVERTIME
 16Ø REM  T3              TOTAL REGULAR PAY
 17Ø REM  T4              TOTAL OVERTIME PAY
*18Ø REM  T5              TOTAL BONUS PAY
 19Ø REM  T6              TOTAL PAYROLL
 2ØØ REM  M$              RESPONSE TO
                          CONTINUE LOOP


 21Ø  LET T1 = Ø

 22Ø  LET T2 = Ø

 23Ø  LET T3 = Ø

 24Ø  LET T4 = Ø

*25Ø  LET T5 = Ø

 26Ø  LET T6 = Ø

 27Ø  PRINT "ENTER DATE"

 28Ø  INPUT D$
```

```
  29Ø  CLS
  3ØØ  PRINT ‘‘ENTER NAME’’
  31Ø  INPUT N$
  32Ø  PRINT ‘‘ENTER HOURS’’
  33Ø  INPUT H
  34Ø  PRINT ‘‘ENTER RATE’’
  35Ø  INPUT R
 *36Ø  PRINT ‘‘ENTER BONUS CODE’’
 *37Ø  INPUT B$
  38Ø  IF H>4Ø THEN GOTO 43Ø
  39Ø  LET A = H * R
  4ØØ  LET B = Ø
  41Ø  LET T1 = T1 + 1
  42Ø  GOTO 46Ø
  43Ø  LET A = R * 4Ø
  44Ø  LET B = (H - 4Ø) * R * 1.5
  45Ø  LET T2 = T2 + 1
  46Ø  LET T3 = T3 + A
  47Ø  LET T4 = T4 + B
 *48Ø  IF B$ = ‘‘A’’ THEN LET C = 2Ø.ØØ
 *49Ø  IF B$ = ‘‘B’’ THEN LET C = 25.ØØ
 *5ØØ  IF B$ = ‘‘C’’ THEN LET C = 3Ø.ØØ
 *51Ø  LET T5 = T5 + C
**52Ø  LET D = A + B + C
  53Ø  LET T6 = T6 + D
  54Ø  CLS
```

**NOTE: ‘‘+C’’ is added to this line.

```
 55Ø  PRINT “NAME”, N$
 56Ø  PRINT “REGULAR PAY”, A
 57Ø  PRINT “OVERTIME PAY”, B
*58Ø  PRINT “BONUS PAY”, C
 59Ø  PRINT “TOTAL PAY”, D
 6ØØ  PRINT “ANY MORE (Y/N)?”
 61Ø  INPUT M$
 62Ø  IF M$ = “Y” THEN GOTO 29Ø
 63Ø  CLS
 64Ø  PRINT “TOTALS FOR”, D$
 65Ø  PRINT “TOTAL REGULAR”, T3
 66Ø  PRINT “TOTAL OVERTIME”,
     T4
*67Ø  PRINT “TOTAL BONUS”, T5
 68Ø  PRINT “TOTAL PAYROLL”, T6
 69Ø  PRINT “EMPS. RECEIVING
     REGULAR, ONLY ”; T1
 7ØØ  PRINT “EMPS. RECEIVING
      OVERTIME”; T2
 71Ø  STOP
```

Add these new codes to the original program (A, B and C) and see how your new payroll application works now. It would also be good practice to do a flowchart of your modified program including these new lines.

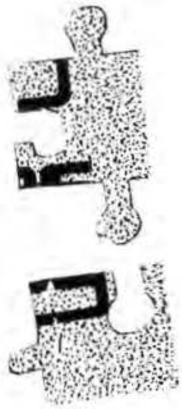
DO YOU KNOW NOW?

These were the questions posed at the beginning of the lesson.

- **The difference between a character and a numeric comparison?**
  Although both comparisons utilize the IF statement, they work quite differently. In a numeric comparison, the fields are aligned on the decimal point and both fields are padded with zeros, if necessary. In an alphanumeric compare, the fields are made equal by padding the shorter field with blanks on the right. The comparison then takes place byte-by-byte from left to right. Values are assigned to characters by the collating sequence.

- **The way conditional branching is accomplished in a one-dimensional program?**
  Branching can be inserted via conditional IF statements in which GOTO statements will be executed only when the IF statement is true.

- **How AND and OR can be used to make a complex IF statement?**
  AND and OR cause more than one condition to be checked in the same IF statements. If AND is used, both conditions must be true for the THEN clause to be executed; if OR is used, only one of the conditions needs to be true.



SCHOOL OF COMPUTER TRAINING

EXAM 7

COMPUTER LOGIC — THE LIMITS OF COMPUTER "INTELLIGENCE"

24707-2

Questions 1-20: *Circle the letter beside the one best answer to each question*

1. Computer "intelligence" is usually described by the ability of the computer to

(a) compare two values.	(c) make up a payroll.
(b) identify colors.	(d) expand memory.

2. If A>B, then...

(a) A is a larger quantity than B.
(b) B is a larger quantity than A.
(c) A is less than or equal to B.
(d) B is greater than or equal to A.

3. In order to compare two numeric values, the CPU must always do two preliminary steps:

(a) Align the decimal points and make the fields equal in length by padding one field or the other field or both.
(b) Pad at least one of the fields and place one decimal point in each field.
(c) Place the decimal point at the same digit in the second field as found in the first.
(d) Add decimal points and zeros so the two numbers are of identical value.

4. Whenever possible, the statement which should be at the very bottom of your flowchart is

(a) your name.
(b) the name of the program.
(c) STOP or END.
(d) GOTO.

5. In BASIC, the keyword used to make a comparison is

(a) GREATER.
(b) THAN.
(c) IF.
(d) GOTO.

6. When the line: 1Ø IF X < Y THEN GOTO 4Ø is entered and run, equal values

(a) cause a branch to line 4Ø.
(b) will not cause a branch.
(c) cause the line to loop continuously.
(d) cause the program to pause for more input.

7. The "collating sequence" refers to

(a) the computer's ability to sort addresses for mail order.
(b) the programmer's ability to produce loops.
(c) the assignment of LET statements.
(d) the computer's built-in assignment of values to every character.

8. When making alphanumeric comparisons, the results of using EBCDIC and ASCII systems

(a) will sometimes be exactly opposite.
(b) will always be the same.
(c) are regulated by law.
(d) are consistently inaccurate.

9. When string values are compared by the computer,

(a) the fields are compared byte by byte from left to right
(b) the bytes are compared field by field from left to right.
(c) the fields are compared byte by byte from right to left.
(d) the bytes are compared from right to left.

10. When comparing numeric values, it is essential that

(a) the numbers end in zero.
(b) the fields are defined as strings.
(c) the fields are defined in alpha order.
(d) the fields are defined as numeric.

11. String variables may be compared to constants as long as

(a) the constant does not exceed five digits.
(b) the constant does not exceed eight bits.
(c) the constant is enclosed within quotation marks.
(d) the constant is also a variable.

12. The logical operators AND and OR can be used

(a) to make singular comparisons.
(b) to make unequal comparisons only.
(c) to make compound IF statements.
(d) to make singular LET statements.

13. When making alphanumeric comparisons, fields are made equal by

(a) padding the longer field with zeros on the left.
(b) padding the shorter field with blanks on the left.
(c) padding the longer field with zeros on the right.
(d) padding the shorter field with blanks on the right.

14. When making a complex IF statement and AND is used,

(a) both conditions must be true for the THEN clause to be executed.
(b) one of the conditions can be false and the THEN clause will be executed.
(c) neither condition must be true for the THEN clause to be executed.
(d) the result will be false because AND should not be used with IF.

15. In a numeric comparison, the fields are

(a) aligned on the comma and both fields are padded with zeros, if necessary.
(b) aligned on the right and the shorter field is padded with zeros, if necessary.
(c) aligned on the decimal point and both fields are padded with zeros, if necessary.
(d) aligned on the decimal point and the shorter field is padded with a comma, if necessary.

16. The second part of the IF statement is

(a) the GOTO statement.
(b) the THEN clause.
(c) the LET clause.
(d) the GREATER THAN statement.

17. The lowest "printable" character in the collating sequence of the EBCDIC system is

(a) the letter "a."
(b) the comma.
(c) a space or blank.
(d) an asterisk.

18. In terms of collating sequence values in the EBCDIC system,

(a) numbers are greater than letters.
(b) numbers have equal value to letters.
(c) numbers have less value than letters.
(d) numbers are not included in the collating sequence.

19. The further the letter is within the alphabet,

(a) the less value it has.
(b) the greater its value.
(c) the less it is used, because all letters are of equal value.
(d) the fewer numbers can be used with it.

20. When an unequal pair of bytes is encountered during a comparison of string values,

(a) the comparison immediately ends and the field with the greater values in that byte is said to be greater.
(b) the comparison immediately ends and the field with the lesser values in that byte is said to be greater.
(c) the comparison continues until another byte is found which is larger.
(d) the comparison immediately ends and the first byte is said to be greater than the second byte.

WHEN YOU HAVE COMPLETED THE ENTIRE EXAM, TRANSFER YOUR ANSWERS TO THE ANSWER SHEET WHICH FOLLOWS.

ICS

ANSWER PAPER

To avoid delay, please insert all the details requested below

Subject PRACTICAL PROGRAMMING IN BASIC Course

Name

Address

Post Code

Serial Number	Test No.	Ed No.
2 4 7 0 7	7	2

Student's Reference

Letters / Figures

Tutor's Comments

Grade	Tutor

Study the foregoing Question Paper and use it for your rough workings. Record your final answers in the matrix below by writing a cross (X), IN INK OR BALLPOINT, through the letter which you think is the correct answer. Submit ONLY THIS ANSWER SHEET to the School for correction. ALL QUESTIONS MUST BE ANSWERED.

1.	A	B	C	D
2.	A	B	C	D
3.	A	B	C	D
4.	A	B	C	D
5.	A	B	C	D

6.	A	B	C	D
7.	A	B	C	D
8.	A	B	C	D
9.	A	B	C	D
10.	A	B	C	D

11.	A	B	C	D
12.	A	B	C	D
13.	A	B	C	D
14.	A	B	C	D
15.	A	B	C	D

16.	A	B	C	D
17.	A	B	C	D
18.	A	B	C	D
19.	A	B	C	D
20.	A	B	C	D